

Muffix Sassif – TRD

Andrianov, Lepeshov, Shulyatev

December 13, 2025



Содержание

1	Геометрия	2	4	Деревья	12	7	Строки	22
1.1	3D	2	4.1	Centroid	12	7.1	Z-функция	22
1.2	База 1 - вектор	2	4.2	HLD	12	7.2	eertree	22
1.3	База 2 - прямая	3	4.3	Link-cut	13	7.3	Ахо-Корасик	22
1.4	База 3 - окружность	3				7.4	Муффиксный Сассив	23
1.5	Выпуклая оболочка	3	5	Другое	14	7.5	Префикс-функция	23
1.6	Задача 16	3	5.1	Fast mod	14	7.6	Суффиксный автомат	23
1.7	Калиперы	4	5.2	Slope trick	14	8	Структуры данных	23
1.8	Касательные из точки	4	5.3	attribute_packed	14	8.1	Disjoint Sparse Table	23
1.9	Касательные параллельные прямой	4	5.4	custom_bitset	14	8.2	Segment Tree Beats	24
1.10	Лежит ли точка в многоугольнике	4	5.5	ordered_set	14	8.3	ДД по неявному	24
1.11	Минимальная покрывающая окружность	5	5.6	pragma	15	8.4	ДД	24
1.12	Пересечение полуплоскостей	5	5.7	Аллокатор Копелиовича	15	8.5	Персистентное ДД по неявному	25
1.13	Проверка на пересечение отрезков	5	5.8	Альфа-бета отсечение	15	8.6	Персистентное ДО	25
1.14	Сумма Минковского	5	5.9	Отжиг	15	8.7	Спарсы	26
1.15	Формула Эйлера	6	6	Математика	15	8.8	Фенвик (+ на отрезке)	26
2	Графы	6	6.1	AdivB cmp CdivD	15	8.9	Фенвик	26
2.1	2-SAT	6	6.2	Berlecamp	15			
2.2	L-R Flow	6	6.3	FFT mod	16			
2.3	WeightedMatching	6	6.4	FFT	17			
2.4	Венгерский алгоритм	8	6.5	Floor Sum	17			
2.5	Вершинная двусвязность	8	6.6	GCD LCM свёртки	17			

1 Геометрия

1.1 3D

```
struct Pt {
    dbl x, y, z;

    Pt() : x(0), y(0), z(0) {}
    Pt(dbl x_, dbl y_, dbl z_) : x(x_), y(y_), z(z_) {}

    Pt operator-(const Pt& o) const {
        return {x - o.x, y - o.y, z - o.z};
    }
    Pt operator+(const Pt& o) const {
        return {x + o.x, y + o.y, z + o.z};
    }
    Pt operator/(const dbl& a) const {
        return {x / a, y / a, z / a};
    }
    Pt operator*(const dbl& a) const {
        return {x * a, y * a, z * a};
    }
    Pt cross(const Pt& o) const {
        dbl nx = y * o.z - z * o.y;
        dbl ny = z * o.x - x * o.z;
        dbl nz = x * o.y - y * o.x;
        return {nx, ny, nz};
    }
    dbl dot(const Pt &o) const {
        return x * o.x + y * o.y + z * o.z;
    }
    bool operator==(const Pt& o) const {
        return abs(x - o.x) < EPS && abs(y - o.y) < EPS &&
            abs(z - o.z) < EPS;
    }
    dbl dist() {
        return sqrtl(x * x + y * y + z * z);
    }
};

struct Plane {
    dbl a, b, c, d;

    Plane(dbl a_, dbl b_, dbl c_, dbl d_) : a(a_), b(b_),
        c(c_), d(d_) {
        dbl z = sqrtl(a * a + b * b + c * c);
        if (z < EPS) return;
        a /= z, b /= z, c /= z, d /= z;
    }
    dbl get_val(const Pt &p) const {
        // НЕ СТАВИТЬ МОДУЛЬ
        return a * p.x + b * p.y + c * p.z + d;
    }
    dbl dist(const Pt &p) const {
        return abs(get_val(p));
    }
    bool on_plane(const Pt &p) const {
        return dist(p) / sqrtl(a * a + b * b + c * c) < EPS;
    }
    Pt proj(const Pt &p) const {
```

```
        dbl t = get_val(p) / (a * a + b * b + c * c);
        return p - Pt(a, b, c) * t;
    }
};

bool on_line(Pt p1, Pt p2, Pt p3) {
    return (p2 - p1).cross(p3 - p1) == Pt(0, 0, 0);
}

Plane get_plane(Pt p1, Pt p2, Pt p3) {
    Pt norm = (p2 - p1).cross(p3 - p1);
    Plane pl(norm.x, norm.y, norm.z, 0);
    pl.d = -pl.get_val(p1);
    return pl;
}

pair<pair<dbl, dbl>, pair<dbl, dbl>> get_xy(dbl a, dbl b
    , dbl c) {
    if (abs(a) > EPS) {
        dbl y1 = 0, y2 = 10;
        return {{(-c - b * y1) / a, y1}, {(-c - b * y2) / a,
            y2}};
    }
    dbl x1 = 0, x2 = 10;
    return {{x1, (-c - a * x1) / b}, {x2, (-c - a * x2) /
        b}};
}

pair<Pt, Pt> intersect(Plane pl1, Plane pl2) {
    if (abs(pl2.a) < EPS && abs(pl2.b) < EPS && abs(pl2.c)
        < EPS) {
        assert(false);
    }
    if (abs(pl2.a) > EPS) {
        dbl nd = pl1.d - pl1.a * pl2.d / pl2.a;
        dbl nc = pl1.c - pl1.a * pl2.c / pl2.a;
        dbl nb = pl1.b - pl1.a * pl2.b / pl2.a;
        if (abs(nc) < EPS && abs(nb) < EPS) {
            // плоскости параллельны (могут совпадать)
            return {Pt(0, 0, 0), Pt(0, 0, 0)};
        }
        auto [yz1, yz2] = get_xy(nb, nc, nd);
        dbl x1 = (-pl2.d - pl2.c * yz1.second - pl2.b * yz1.
            first) / pl2.a;
        dbl x2 = (-pl2.d - pl2.c * yz2.second - pl2.b * yz2.
            first) / pl2.a;
        return {Pt(x1, yz1.first, yz1.second), Pt(x2, yz2.
            first, yz2.second)};
    }
    Plane copy_pl1(pl1.c, pl1.a, pl1.b, pl1.d);
    Plane copy_pl2(pl2.c, pl2.a, pl2.b, pl2.d);
    auto [p1, p2] = intersect(copy_pl1, copy_pl2);
    return {Pt(p1.y, p1.z, p1.x), Pt(p2.y, p2.z, p2.x)};
}

// угол между двумя векторами
dbl get_ang(Pt p1, Pt p2) {
    return acosl(p1.dot(p2) / p1.dist() / p2.dist());
}
```

```
// любой перпендикулярный вектор
Pt vector_perp(Pt v) {
    if (abs(v.x) > EPS || abs(v.y) > EPS)
        return {v.y, -v.x, 0};
    return {v.z, 0, -v.x};
}

// плоскость через точку p перпендикулярная вектору v
Plane plane_perp(Pt p, Pt v) {
    Pt v1 = vector_perp(v);
    Pt v2 = v.cross(v1);
    return get_plane(p, v1 + p, v2 + p);
}
```

1.2 База 1 - вектор

```
char sign(dbl x) {
    return x < -EPS ? -1 : x > EPS;
}

struct vctr {
    dbl x, y;
    vctr() {}
    vctr(dbl x, dbl y) : x(x), y(y) {}

    dbl operator%(const vctr &o) const { return x * o.x +
        y * o.y; }
    dbl operator*(const vctr &o) const { return x * o.y -
        y * o.x; }
    vctr operator+(const vctr &o) const { return {x + o.x,
        y + o.y}; }
    vctr operator-(const vctr &o) const { return {x - o.x,
        y - o.y}; }
    vctr operator-() const { return {-x, -y}; }
    vctr operator*(const dbl d) const { return {x * d, y *
        d}; }
    vctr operator/(const dbl d) const { return {x / d, y /
        d}; }
    void operator+=(const vctr &o) { x += o.x, y += o.y; }
    void operator-=(const vctr &o) { x -= o.x, y -= o.y; }
    dbl dist2() const { return x * x + y * y; }
    dbl dist() const { return sqrtl(dist2()); }
    vctr norm() const { return *this / dist(); }
};

dbl angle_between(const vctr &a, const vctr &b) {
    return atan2(b * a, b % a);
}

// y > 0 ? 0 : 1
bool is2plane(const vctr &a) {
    return sign(a.y) < 0 || (sign(a.y) == 0 && sign(a.x) <
        0);
}

bool cmp_angle(const vctr &a, const vctr &b) {
    bool pla = is2plane(a);
    bool plb = is2plane(b);
    if (pla != plb)
```

```

    return pla < plb;
    return sign(a * b) > 0;
}

vctr rotate_ccw(const vctr &a, dbl phi) {
    dbl cs = cos(phi);
    dbl sn = sin(phi);
    return {a.x * cs - a.y * sn, a.y * cs + a.x * sn};
}

vctr rotate_ccw_90(const vctr &a) {
    return {-a.y, a.x};
}

```

1.3 База 2 - прямая

```

struct line {
    dbl a, b, c;

    line() {}
    line(dbl a, dbl b, dbl c) : a(a), b(b), c(c) {}
    line(const vctr A, const vctr B) {
        a = A.y - B.y;
        b = B.x - A.x;
        c = A * B;
        // left halfplane of A->B is positive
        // assert(a != 0 || b != 0);
    }
    void operator*=(dbl x) { a *= x, b *= x, c *= x; }
    void operator/=(dbl x) { a /= x, b /= x, c /= x; }
    dbl get(const vctr P) const { return a * P.x + b * P.y + c; }
    vctr anyPoint() const {
        dbl x = -a * c / (a * a + b * b);
        dbl y = -b * c / (a * a + b * b);
        return {x, y};
    }
    void normalize() {
        dbl d = sqrtl(a * a + b * b);
        a /= d, b /= d, c /= d;
    }
};

bool isparallel(line l1, line l2) {
    return sign(l2.a * l1.b - l2.b * l1.a) == 0;
}

vctr intersection(const line &l1, const line &l2) {
    dbl z = l2.a * l1.b - l2.b * l1.a;
    dbl x = (l1.c * l2.b - l2.c * l1.b) / z;
    dbl y = -(l1.c * l2.a - l2.c * l1.a) / z;
    return {x, y};
}

// Серединный перпендикуляр (не биссектриса!)
line bisection(const vctr A, const vctr B) {
    vctr M = (A + B) / 2;
    return line(M, M + rotate_ccw_90(B - A));
}

```

1.4 База 3 - окружность

```

struct circle {
    vctr C;
    dbl r;

    circle() {}
    circle(dbl x, dbl y, dbl r) : C(x, y), r(r) {}
    circle(vctr C, dbl r) : C(C), r(r) {}
    circle(const vctr A, const vctr B) {
        C = (A + B) / 2;
        r = (A - B).dist() / 2;
    }
    circle(const vctr A, const vctr B, const vctr D) {
        line l1 = bisection(A, B);
        line l2 = bisection(B, D);
        C = intersection(l1, l2);
        r = (C - A).dist();
    }

    bool isin(const vctr P) const {
        return sign((C - P).dist2() - r * r) <= 0;
    }
};

vector<vctr> intersection_line_circ(line l, circle c) {
    l.normalize();
    dbl d = abs(l.get(c.C));
    vctr per = vctr(l.a, l.b).norm() * d;
    vctr a = c.C + per;
    if (sign(d - c.r) > 0)
        return {};
    if (sign(l.get(a)) != 0)
        a = c.C - per;
    if (sign(c.r - d) == 0)
        return {a};
    dbl k = sqrtl(c.r * c.r - d * d);
    vctr v = vctr(-l.b, l.a).norm() * k;
    return {a + v, a - v};
}

vector<vctr> intersection_circ_circ(circle A, circle B) {
    {
        vctr a = A.C, b = B.C;
        line l(2 * (b.x - a.x),
              2 * (b.y - a.y),
              B.r * B.r - A.r * A.r
              + (a.x * a.x + a.y * a.y)
              - (b.x * b.x + b.y * b.y));
        if (sign(l.a) == 0 && sign(l.b) == 0)
            return {};
        return intersection_line_circ(l, A);
    }
}

vector<vctr> tangent_vctr_circ(vctr v, circle c) {
    dbl d = (c.C - v).dist();
    dbl k = sqrtl(d * d - c.r * c.r);
}

```

```

    circle c2(v.x, v.y, k);
    return intersection_circ_circ(c, c2);
}

```

1.5 Выпуклая оболочка

```

vctr minvctr(INF, INF);

bool cmp_convex_hull(const vctr &a, const vctr &b) {
    vctr A = a - minvctr;
    vctr B = b - minvctr;
    auto sign_prod = sign(A * B);
    if (sign_prod != 0)
        return sign_prod > 0;
    return A.dist2() < B.dist2();
}

// minvctr updates here
vector<vctr> get_convex_hull(vector<vctr> arr) {
    minvctr = {INF, INF};
    for (auto v : arr) {
        auto tmp = v - minvctr;
        if (sign(tmp.y) < 0 || (sign(tmp.y) == 0 && sign(tmp.x) < 0))
            minvctr = v;
    }
    vector<vctr> hull;
    sort(arr.begin(), arr.end(), cmp_convex_hull);
    for (vctr &el : arr) {
        while (hull.size() > 1 && sign((hull.back() - hull[hull.size() - 2]) * (el - hull.back())) <= 0)
            hull.pop_back();
        hull.push_back(el);
    }
    return hull;
}

1.6 Задача 16

bool isInSameHalf(vctr p, vctr r1, vctr r2) {
    return sign((r2 - r1) % (p - r1)) >= 0;
}

dbl distPointPoint(vctr a, vctr b) {
    return (a - b).dist();
}

dbl distPointLine(vctr a, vctr l1, vctr l2) {
    line l(l1, l2);
    l.normalize();
    return abs(l.get(a));
}

dbl distPointRay(vctr a, vctr r1, vctr r2) {
    if (!isInSameHalf(a, r1, r2))
        return distPointPoint(a, r1);
    return distPointLine(a, r1, r2);
}

```

```

}

dbl distPointSeg(vctr a, vctr s1, vctr s2) {
    return max(distPointRay(a, s1, s2),
               distPointRay(a, s2, s1));
}

bool isIntersectionLineLine(line l1, line l2) {
    dbl znam = l1.b * l2.a - l1.a * l2.b;
    return sign(znam) != 0;
}

vctr intersectionLineLine(line l1, line l2) {
    dbl znam = l1.b * l2.a - l1.a * l2.b;
    dbl y = -(l1.c * l2.a - l2.c * l1.a) / znam;
    dbl x = -(l1.c * l2.b - l2.c * l1.b) / -znam;
    return vctr(x, y);
}

vctr getPointOnLine(line l) {
    if (sign(l.b) != 0)
        return vctr(0, -l.c / l.b);
    return vctr(-l.c / l.a, 0);
}

dbl distLineLine(vctr l1a, vctr l1b, vctr l2a, vctr l2b) {
    line l1(l1a, l1b);
    line l2(l2a, l2b);
    if (isIntersectionLineLine(l1, l2))
        return 0;
    vctr p = getPointOnLine(l1);
    l2.normalize();
    return abs(l2.get(p));
}

dbl distRayLine(vctr r1, vctr r2, vctr l1, vctr l2) {
    line r(r1, r2);
    line l(l1, l2);
    if (!isIntersectionLineLine(l, r))
        return distLineLine(r1, r2, l1, l2);
    vctr p = intersectionLineLine(l, r);
    if (isInSameHalf(p, r1, r2))
        return 0;
    return distPointLine(r1, l1, l2);
}

dbl distSegLine(vctr s1, vctr s2, vctr l1, vctr l2) {
    return max(distRayLine(s1, s2, l1, l2),
               distRayLine(s2, s1, l1, l2));
}

dbl distRayRay(vctr r1a, vctr r1b, vctr r2a, vctr r2b) {
    line r1(r1a, r1b);
    line r2(r2a, r2b);
    if (!isIntersectionLineLine(r1, r2)) {
        if (isInSameHalf(r1a, r2a, r2b) || isInSameHalf(r2a,
                                                         r1a, r1b))
            return distLineLine(r1a, r1b, r2a, r2b);
        else

```

```

        return distPointPoint(r1a, r2a);
    }
    vctr p = intersectionLineLine(r1, r2);
    if (isInSameHalf(p, r1a, r1b) && isInSameHalf(p, r2a,
                                                  r2b))
        return 0;
    return min(distPointRay(r1a, r2a, r2b),
               distPointRay(r2a, r1a, r1b));
}

dbl distSegRay(vctr s1, vctr s2, vctr r1, vctr r2) {
    return max(distRayRay(s1, s2, r1, r2),
               distRayRay(s2, s1, r1, r2));
}

dbl distSegSeg(vctr s1a, vctr s1b, vctr s2a, vctr s2b) {
    return max(distSegRay(s1a, s1b, s2a, s2b),
               distSegRay(s1a, s1b, s2b, s2a));
}

```

1.7 Калиперы

```

// Диаметр выпуклого многоугольника
int calipers(vector<vctr> &pts) {
    int n = pts.size();
    int a = 0, b = 0;
    for (int i = 1; i < n; ++i) {
        auto &v = pts[i];
        if (tie(v.y, v.x) < tie(pts[a].y, pts[a].x))
            a = i;
        if (tie(v.y, v.x) > tie(pts[b].y, pts[b].x))
            b = i;
    }
    int aa = (a + 1) % n, bb = (b + 1) % n;
    int dist2 = 0;
    for (int i = 0; i < n; ++i) {
        while (sign((pts[aa] - pts[a]) * (pts[bb] - pts[b]))
              > 0)
            b = bb, bb = (b + 1) % n;
        dist2 = max(dist2, (pts[a] - pts[b]).dist2());
        a = aa, aa = (a + 1) % n;
    }
    return dist2;
}

```

1.8 Касательные из точки

```

pair<int, int> tangents_from_point(vector<vctr> &p, vctr
&a) {
    int n = p.size();
    int logn = 31 - __builtin_clz(n);
    auto findWithSign = [&](int sgn) {
        int i = 0;
        for (int k = logn; k >= 0; --k) {
            int i1 = (i - (1 << k) + n) % n;
            int i2 = (i + (1 << k)) % n;
            if (sign((p[i1] - a) * (p[i] - a)) == sgn)

```

```

            i = i1;
            if (sign((p[i2] - a) * (p[i] - a)) == sgn)
                i = i2;
        }
        return i;
    };
    return {findWithSign(1), findWithSign(-1)};
}

```

1.9 Касательные параллельные прямой

```

// find point with max (sgn=1) or min (sgn=-1) signed
// distance to line
int tangent_parallel_line(const vector<vctr> &p, line l,
int sgn) {
    l *= sgn;
    int n = p.size();
    int i = 0;
    int logn = 31 - __builtin_clz(n);
    for (int k = logn; k >= 0; --k) {
        int i1 = (i - (1 << k) + n) % n;
        int i2 = (i + (1 << k)) % n;
        if (l.get(p[i1]) > l.get(p[i]))
            i = i1;
        if (l.get(p[i2]) > l.get(p[i]))
            i = i2;
    }
    return i;
}

```

1.10 Лежит ли точка в многоугольнике

```

// Выпуклый многоугольник, P[0] = minvctr
bool is_point_in_poly(vctr A, vector<vctr> &P) {
    auto tmp = A - P[0];
    if (sign(tmp.y) < 0 || (sign(tmp.y) == 0 && sign(tmp.x)
                          < 0))
        return false;
    if (sign(tmp.y) == 0 && sign(tmp.x) == 0)
        return true;
    int ind = lower_bound(P.begin(), P.end(), A,
                          cmp_convex_hull) - P.begin();
    assert(ind != 0);
    if (ind == P.size())
        return false;
    vctr B = A - P[ind - 1];
    vctr C = P[ind] - P[ind - 1];
    return sign(C * B) >= 0;
}

bool is_point_in_poly_strict(vctr A, vector<vctr> &P) {
    if (sign(A.y - P[0].y) <= 0 || sign((A - P[0]) * (P.
        back() - P[0])) <= 0)
        return false;
    int ind = lower_bound(P.begin(), P.end(), A,
                          cmp_convex_hull) - P.begin();
    assert(ind != 0 && ind != P.size());
}

```

```

    vctr B = A - P[ind - 1];
    vctr C = P[ind] - P[ind - 1];
    return sign(C * B) > 0;
}

```

1.11 Минимальная покрывающая окружность

```
mt19937 rnd(179);
```

```

circle MinDisk2(vector<vctr> &p, vctr A, vctr B, int sz)
{
    circle w(A, B);
    for (int i = 0; i < sz; ++i) {
        if (w.isin(p[i]))
            continue;
        w = circle(A, B, p[i]);
    }
    return w;
}

```

```

circle MinDisk1(vector<vctr> &p, vctr A, int sz) {
    shuffle(p.begin(), p.begin() + sz, rnd);
    circle w(A, p[0]);
    for (int i = 1; i < sz; ++i) {
        if (w.isin(p[i]))
            continue;
        w = MinDisk2(p, A, p[i], i);
    }
    return w;
}

```

```

circle MinDisk(vector<vctr> &p) {
    int sz = p.size();
    if (sz == 1)
        return circle(p[0], 0);
    shuffle(p.begin(), p.end(), rnd);
    circle w(p[0], p[1]);
    for (int i = 2; i < sz; ++i) {
        if (w.isin(p[i]))
            continue;
        w = MinDisk1(p, p[i], i);
    }
    return w;
}

```

1.12 Пересечение полуплоскостей

```

// half plane: ax+by+c > 0
// bounding box MUST have
vector<int> intersection_half_planes_inds(const vector<
    line> &ls) {
    int n = (int)ls.size();
    vector<int> lsi(n);
    iota(lsi.begin(), lsi.end(), 0);
    sort(lsi.begin(), lsi.end(), [&](int i, int j) {
        vctr aa(ls[i].a, ls[i].b);
        vctr bb(ls[j].a, ls[j].b);

```

```

        bool pla = is2plane(aa);
        bool plb = is2plane(bb);
        if (pla != plb)
            return pla < plb;
        return aa * bb > 0;
    });

    vector<line> st;
    vector<int> inds;
    for (int ii = 0; ii < 2 * n; ++ii) {
        int i = lsi[ii % n];
        if (st.empty()) {
            st.push_back(ls[i]);
            inds.push_back(i);
            continue;
        }
        vctr p = intersection(ls[i], st.back());
        bool pp = isparallel(ls[i], st.back());
        bool bad = false;
        while (st.size() >= 2) {
            if (!pp && sign(st[st.size() - 2].get(p)) >= 0)
                break;
            else if (pp && sign(st.back().get(ls[i].anyPoint())
                <= 0) {
                bad = true;
                break;
            }
            st.pop_back();
            inds.pop_back();
            p = intersection(ls[i], st.back());
            pp = isparallel(ls[i], st.back());
        }
        if (!bad) {
            st.push_back(ls[i]);
            inds.push_back(i);
        }
    }
    vector<int> cnt(n, 0);
    for (int i : inds)
        cnt[i]++;
    vector<int> good;
    for (int i : inds) {
        if (cnt[i]-- == 2)
            good.push_back(i);
    }
    return good;
}

```

```

vector<vctr> intersection_half_planes(vector<line> &ls)
{
    vector<int> inter = intersection_half_planes_inds(ls);
    int n = inter.size();
    vector<vctr> pts;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        vctr P = intersection(ls[inter[i]], ls[inter[j]]);
        if (pts.empty() || sign(pts.back().x - P.x) != 0
            || sign(pts.back().y - P.y) != 0)
            pts.push_back(P);
    }
}

```

```

// pts против часовой стрелки, но pts[0] != minvctr
return pts;
}

```

1.13 Проверка на пересечение отрезков

```

bool is_intersection_seg(vctr A, vctr B, vctr C, vctr D)
{
    for (int i = 0; i < 2; ++i) {
        auto l1 = A.x, r1 = B.x, l2 = C.x, r2 = D.x;
        if (l1 > r1) swap(l1, r1);
        if (l2 > r2) swap(l2, r2);
        if (max(l1, l2) > min(r1, r2))
            return false;
        swap(A.x, A.y);
        swap(B.x, B.y);
        swap(C.x, C.y);
        swap(D.x, D.y);
    }
    for (int _ = 0; _ < 2; ++_) {
        auto v1 = (B - A) * (C - A);
        auto v2 = (B - A) * (D - A);
        if (sign(v1) * sign(v2) == 1)
            return false;
        swap(A, C);
        swap(B, D);
    }
    return true;
}

```

1.14 Сумма Минковского

```

// Список вершин -> список рёбер
vector<vctr> poly_to_edges(const vector<vctr> &A) {
    vector<vctr> edg(A.size());
    for (int i = 0; i < A.size(); ++i)
        edg[i] = A[(i + 1) % A.size()] - A[i];
    return edg;
}

// A и B начинаются с минимальных вершин
vector<vctr> minkowski_sum(const vector<vctr> &A, const
    vector<vctr> &B) {
    auto edgA = poly_to_edges(A);
    auto edgB = poly_to_edges(B);
    vector<vctr> edgC(A.size() + B.size());
    merge(edgA.begin(), edgA.end(), edgB.begin(), edgB.end()
        (), edgC.begin(), cmp_angle);
    // cmp_angle из шаблона вектора
    vector<vctr> C(edgC.size());
    C[0] = A[0] + B[0];
    for (int i = 0; i + 1 < C.size(); ++i)
        C[i + 1] = C[i] + edgC[i];
    // могут быть точки на сторонах
    return C;
}

```


1.15 Формула Эйлера

- V – число вершин выпуклого многогранника (планарного графа)
- E – число рёбер
- F – число граней (если планарный граф, то включая внешнюю)

Тогда $V - E + F = 2$

2 Графы

2.1 2-SAT

```
struct TwoSat {
    int n;
    vector<vector<int>> g, rg;
    vector<int> comp, topsort;
    vector<char> used;

    TwoSat(int n) : n(n) {
        g.resize(2 * n);
        rg.resize(2 * n);
        comp.assign(2 * n, -1);
        topsort.reserve(2 * n);
        used.assign(2 * n, 0);
    }

    int neg(int v) {
        return 2 * n - 1 - v;
    }

    void add(int v, int u) {
        g[v].pb(u);
        rg[u].pb(v);
    }

    void add_OR(int v, int u) { // v | u
        add(neg(v), u); add(neg(u), v);
    }

    void add_IMPL(int v, int u) { // v -> u
        add_OR(neg(v), u);
    }

    void dfs1(int v) {
        used[v] = 1;
        for (auto u: g[v]) {
            if (!used[u])
                dfs1(u);
        }
        topsort.push_back(v);
    }
};
```

```
void dfs2(int v, int col) {
    comp[v] = col;
    for (auto u: rg[v]) {
        if (comp[u] == -1)
            dfs2(u, col);
    }
}

void SCC() {
    for (int v = 0; v < 2 * n; ++v) {
        if (!used[v])
            dfs1(v);
    }
    reverse(all(topsort));
    int cc = 0;
    for (int i = 0; i < 2 * n; ++i) {
        if (comp[topsort[i]] == -1)
            dfs2(topsort[i], cc++);
    }
}

vector<char> solution() {
    assert(n > 0); // returns {} if ans exists AND not SCC();
    vector<char> ans(n);
    for (int v = 0; v < n; ++v) {
        if (comp[v] == comp[neg(v)])
            return {}; // no solution
        ans[v] = comp[v] > comp[neg(v)];
    }
    return ans;
};
```

2.2 L-R Flow

```
struct LRFlow {
    Dinic dinic;
    int S, T; // исток и сток
    int Sx, Tx; // вспомогательные вершины, любые неиспользуемые индексы
    vector<ll> dem;

    LRFlow(int n, int S, int T, int Sx, int Tx) : S(S), T(T), Sx(Sx), Tx(Tx), dinic(n), dem(n) {}

    void addedge(int v, int u, int mincap, int maxcap, int i) {
        // i - any number, can be used for flow restoring.
        // Just store it inside Edge.
        dinic.addedge(v, u, maxcap - mincap, i);
        dem[v] -= mincap;
        dem[u] += mincap;
    }

    // returns size of lr-flow (-1 if not exists)
    // edge.real_flow = edge.flow + edge.mincap
    ll run() {
```

```
    dinic.addedge(T, S, INF, -1); // INF >= sum maxcap - mincap
    ll totaldem = 0;
    for (int v = 0; v < dem.size(); ++v) {
        if (dem[v] > 0) {
            totaldem += dem[v];
            // here capacity could potentially be long long
            // (sum of v's mincaps)
            dinic.addedge(Sx, v, dem[v], -1);
        } else if (dem[v] < 0)
            dinic.addedge(v, Tx, -dem[v], -1);
    }
    dinic.run(Sx, Tx);
    ll flow = 0;
    for (auto &e : dinic.graph[Sx]) flow += e.f;
    if (flow != totaldem) return -1;
    for (int v = 0; v < dem.size(); ++v) {
        if (dem[v]) dinic.graph[v].pop_back();
    }
    dinic.graph[Sx].clear();
    dinic.graph[Tx].clear();
    dinic.graph[S].pop_back();
    dinic.graph[T].pop_back();
    // dinic.run(S, T); // if we want max lr-flow (not any lr-flow)
    for (auto &e : dinic.graph[S]) flow += e.f;
    return flow;
};
```

2.3 WeightedMatching

```
// НЕ ЗАБЫТЬ вызвать init(n)
// вершины нумеруются от 1 до n
namespace weighted_matching {
    const int INF = (int)1e9 + 7;
    const int MAXN = 1050; //double of possible N
    struct E {
        int x, y, w;
    };
    int n, m;
    E G[MAXN][MAXN];
    int lab[MAXN], match[MAXN], slack[MAXN], st[MAXN], pa[MAXN];
    int flo_from[MAXN][MAXN], S[MAXN], vis[MAXN];
    vector<int> flo[MAXN];
    queue<int> Q;
    void init(int _n) {
        n = _n;
        for (int x = 1; x <= n; ++x)
            for (int y = 1; y <= n; ++y)
                G[x][y] = E{x, y, 0};
    }
    void add_edge(int x, int y, int w) {
        G[x][y].w = G[y][x].w = w;
    }
    int e_delta(E e) {
        return lab[e.x] + lab[e.y] - G[e.x][e.y].w * 2;
    }
};
```

```

void update_slack(int u, int x) {
    if(!slack[x] || e_delta(G[u][x]) < e_delta(G[slack[x]]
        [x]))
        slack[x] = u;
}
void set_slack(int x) {
    slack[x] = 0;
    for(int u = 1; u <= n; ++u)
        if(G[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
            update_slack(u, x);
}
void q_push(int x) {
    if(x <= n) Q.push(x);
    else for(int i = 0; i < (int)flo[x].size(); ++i)
        q_push(flo[x][i]);
}
void set_st(int x, int b) {
    st[x] = b;
    if(x > n) for(int i = 0; i < (int)flo[x].size(); ++i)
        set_st(flo[x][i], b);
}
int get_pr(int b, int xr) {
    int pr = find(flo[b].begin(), flo[b].end(), xr) -
        flo[b].begin();
    if(pr & 1) {
        reverse(flo[b].begin() + 1, flo[b].end());
        return (int)flo[b].size() - pr;
    }
    else return pr;
}
void set_match(int x, int y) {
    match[x] = G[x][y].y;
    if(x <= n) return;
    E e = G[x][y];
    int xr = flo_from[x][e.x], pr = get_pr(x, xr);
    for(int i = 0; i < pr; ++i) set_match(flo[x][i], flo
        [x][i+1]);
    set_match(xr, y);
    rotate(flo[x].begin(), flo[x].begin() + pr, flo[x].
        end());
}
void augment(int x, int y) {
    while(1) {
        int ny = st[match[x]];
        set_match(x, y);
        if(!ny) return;
        set_match(ny, st[pa[ny]]);
        x = st[pa[ny]], y = ny;
    }
}
int get_lca(int x, int y) {
    static int t = 0;
    for(++t; x || y; swap(x, y)) {
        if(x == 0) continue;
        if(vis[x] == t) return x;
        vis[x] = t;
        x = st[match[x]];
        if(x) x = st[pa[x]];
    }
}

```

```

        return 0;
    }
    void add_blossom(int x, int l, int y) {
        int b = n + 1;
        while(b <= m && st[b]) ++b;
        if(b > m) ++m;
        lab[b] = 0, S[b] = 0;
        match[b] = match[l];
        flo[b].clear();
        flo[b].push_back(l);
        for(int u = x, v; u != 1; u = st[pa[v]])
            flo[b].push_back(u), flo[b].push_back(v = st[match
                [u]]), q_push(v);
        reverse(flo[b].begin() + 1, flo[b].end());
        for(int u = y, v; u != 1; u = st[pa[v]])
            flo[b].push_back(u), flo[b].push_back(v = st[match
                [u]]), q_push(v);
        set_st(b, b);
        for(int i = 1; i <= m; ++i) G[b][i].w = G[i][b].w =
            0;
        for(int i = 1; i <= n; ++i) flo_from[b][i] = 0;
        for(int i = 0; i < (int)flo[b].size(); ++i) {
            int us = flo[b][i];
            for(int u = 1; u <= m; ++u)
                if(G[b][u].w == 0 || e_delta(G[us][u]) < e_delta
                    (G[b][u]))
                    G[b][u] = G[us][u], G[u][b] = G[u][us];
            for(int u = 1; u <= n; ++u)
                if(flo_from[us][u])
                    flo_from[b][u] = us;
        }
        set_slack(b);
    }
    void expand_blossom(int b) {
        for(int i = 0; i < (int)flo[b].size(); ++i)
            set_st(flo[b][i], flo[b][i]);
        int xr = flo_from[b][G[b][pa[b]].x], pr = get_pr(b,
            xr);
        for(int i = 0; i < pr; i += 2) {
            int xs = flo[b][i], xns = flo[b][i + 1];
            pa[xs] = G[xns][xs].x;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for(int i = pr + 1; i < (int)flo[b].size(); ++i) {
            int xs = flo[b][i];
            S[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    bool on_found_edge(E e) {
        int x = st[e.x], y = st[e.y];
        if(S[y] == -1) {
            pa[y] = e.x, S[y] = 1;
            int ny = st[match[y]];
            slack[y] = slack[ny] = 0;
            S[ny] = 0, q_push(ny);
        }
    }
}

```

```

    else if(S[y] == 0) {
        int l = get_lca(x, y);
        if(!l) return augment(x, y), augment(y, x), true;
        else add_blossom(x, l, y);
    }
    return false;
}
bool matching() {
    fill(S + 1, S + m + 1, -1);
    fill(slack + 1, slack + m + 1, 0);
    Q = queue<int>();
    for(int x = 1; x <= m; ++x)
        if(st[x] == x && !match[x]) pa[x] = 0, S[x] = 0,
            q_push(x);
    if(Q.empty()) return false;
    while(1) {
        while(Q.size()) {
            int x = Q.front(); Q.pop();
            if(S[st[x]] == 1) continue;
            for(int y = 1; y <= n; ++y) {
                if(G[x][y].w > 0 && st[x] != st[y]) {
                    if(e_delta(G[x][y]) == 0) {
                        if(on_found_edge(G[x][y])) return true;
                    }
                    else update_slack(x, st[y]);
                }
            }
        }
        int d = INF;
        for(int b = n + 1; b <= m; ++b)
            if(st[b] == b && S[b] == 1) d = min(d, lab[b] /
                2);
        for(int x = 1; x <= m; ++x)
            if(st[x] == x && slack[x]) {
                if(S[x] == -1) d = min(d, e_delta(G[slack[x]]
                    [x]));
                else if(S[x] == 0) d = min(d, e_delta(G[slack
                    [x]][x]) / 2);
            }
        for(int x = 1; x <= n; ++x) {
            if(S[st[x]] == 0) {
                if(lab[x] <= d) return 0;
                lab[x] -= d;
            }
            else if(S[st[x]] == 1) lab[x] += d;
        }
        for(int b = n + 1; b <= m; ++b)
            if(st[b] == b) {
                if(S[st[b]] == 0) lab[b] += d * 2;
                else if(S[st[b]] == 1) lab[b] -= d * 2;
            }
        Q = queue<int>();
        for(int x = 1; x <= m; ++x)
            if(st[x] == x && slack[x] && st[slack[x]] != x
                && e_delta(G[slack[x]][x]) == 0)
                if(on_found_edge(G[slack[x]][x])) return true;
        for(int b = n + 1; b <= m; ++b)
            if(st[b] == b && S[b] == 1 && lab[b] == 0)
                expand_blossom(b);
    }
}

```

```

    return false;
}
pair<int, int> solve(vector<pair<int, int>> &ans) {
    fill(match + 1, match + n + 1, 0);
    m = n;
    int cnt = 0; int sum = 0;
    for(int u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
    int mx = 0;
    for(int x = 1; x <= n; ++x)
        for(int y = 1; y <= n; ++y){
            flo_from[x][y] = (x == y ? x : 0);
            mx = max(mx, G[x][y].w);
        }
    for(int x = 1; x <= n; ++x) lab[x] = mx;
    while(matching()) ++cnt;
    for(int x = 1; x <= n; ++x)
        if(match[x] && match[x] < x) {
            sum += G[x][match[x]].w;
            ans.push_back({x, G[x][match[x]].y});
        }
    return {sum, cnt};
}
}

```

2.4 Венгерский алгоритм

```

pair<int, vector<int>> venger(vector<vector<int>> a) {
    // ищет минимальное по стоимости
    // работает только при n <= m
    // a - массив весов (n+1) × (m+1)
    // a[0][..] = a[..][0] = 0
    // возвращает ans[i] = j если взяли ребро a[i][j]
    int n = (int) a.size() - 1;
    int m = (int) a[0].size() - 1;
    vector<int> u(n + 1), v(m + 1), p(m + 1), way(m + 1);
    for (int i = 1; i <= n; ++i) {
        p[0] = i;
        int j0 = 0;
        vector<int> minv(m + 1, INF);
        vector<char> used(m + 1, false);
        do {
            used[j0] = true;
            int i0 = p[j0], delta = INF, j1;
            for (int j = 1; j <= m; ++j)
                if (!used[j]) {
                    int cur = a[i0][j] - u[i0] - v[j];
                    if (cur < minv[j])
                        minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta)
                        delta = minv[j], j1 = j;
                }
            for (int j = 0; j <= m; ++j)
                if (used[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    minv[j] -= delta;
            j0 = j1;
        } while (p[j0] != 0);
    }
}

```

```

do {
    int j1 = way[j0];
    p[j0] = p[j1];
    j0 = j1;
} while (j0);
}
int cost = -v[0];
vector<int> ans(n + 1);
for (int j = 1; j <= m; ++j)
    ans[p[j]] = j;
return {cost, ans};
}

```

2.5 Вершинная двусвязность

```

vector<pair<int, int>> graph[MAX_V];
bitset<MAX_V> vis;
int st[MAX_E], col[MAX_E], tin[MAX_V], up[MAX_V];
int sti = 0, cc = 0, tt = 0;

```

```

void dfs(int v, int pei) {
    vis[v] = true;
    int upv = tin[v] = tt++;
    for (auto [u, ei] : graph[v]) {
        if (ei == pei) continue;
        if (!vis[u]) {
            int pt = sti;
            st[sti++] = ei;
            dfs(u, ei);
            upv = min(upv, up[u]);
            if (up[u] >= tin[v]) {
                while (sti > pt)
                    col[st[--sti]] = cc;
                cc++;
            }
        } else if (tin[u] <= tin[v]) {
            st[sti++] = ei;
            upv = min(upv, tin[u]);
        }
    }
    up[v] = upv;
}

```

```

// graph[v].emplace_back(u, i);
// graph[u].emplace_back(v, i);
fill(col, col + m, -1);
for (int v = 0; v < n; ++v) {
    if (!vis[v])
        dfs(v, -1);
}
// col[i] - компонента i-го ребра
// cc - итоговое кол-во компонент

```

2.6 Диниц

```

const int LOG = 29; // масштабирование, =0 если не нужно
struct Edge { int u, f, c, r; };

```

```

struct Dinic {
    vector<vector<Edge>> graph;
    vector<char> vis;
    vector<int> inds, dist, Q;
    int ql, qr, S, T, BIT;
    Dinic(int n) : graph(n), vis(n), inds(n), dist(n), Q(n) {}
}

```

```

bool bfs() {
    memset(vis.data(), 0, vis.size());
    ql = 0, qr = 0;
    dist[S] = 0, vis[S] = true;
    Q[qr++] = S;
    while (ql < qr) {
        int v = Q[ql++];
        for (auto &e : graph[v]) {
            int u = e.u;
            if (vis[u] || e.c - e.f < BIT)
                continue;
            vis[u] = true;
            dist[u] = dist[v] + 1;
            Q[qr++] = u;
            if (u == T) return true;
        }
    }
    return false;
}

```

```

int dfs(int v, int maxF) {
    if (v == T) return maxF;
    int ans = 0;
    for (int &i = inds[v]; i < graph[v].size(); ++i) {
        auto &e = graph[v][i];
        auto cc = min(maxF - ans, e.c - e.f);
        if (dist[e.u] <= dist[v] || !vis[e.u] || inds[e.u] == graph[e.u].size() || cc < BIT)
            continue;
        auto f = dfs(e.u, cc);
        if (f != 0) {
            e.f += f, ans += f;
            graph[e.u][e.r].f -= f;
        }
        // иногда быстрее один иф, иногда другой
        if (maxF - ans < 1) break;
        // if (maxF - ans < BIT) break;
    }
    return ans;
}

```

```

void run(int s, int t) {
    S = s, T = t;
    assert(S != T);
    for (BIT = (1ll << LOG); BIT > 0; BIT >= 1) {
        while (bfs()) {
            memset(inds.data(), 0, inds.size() * sizeof(int));
        }
        for (auto &e : graph[S]) {
            if (inds[e.u] == graph[e.u].size() || e.c - e.f < BIT)

```



```

        continue;
        int f = dfs(e.u, e.c - e.f);
        e.f += f, graph[e.u][e.r].f -= f;
    }
}

void addedge(int v, int u, int c) {
    graph[v].push_back({u, 0, c, (int)graph[u].size()});
    // если не ориентированно, то обратная capacity = c
    graph[u].push_back({v, 0, 0, (int)graph[v].size() - 1});
}

};

void use_example() {
    Dinic dinic(n);
    for (int i = 0; i < m; ++i) {
        int v, u, c;
        cin >> v >> u >> c;
        v--, u--;
        dinic.addedge(v, u, c);
    }
    dinic.run(s, t);

    ll maxflow = 0;
    for (auto &e : dinic.graph[s])
        maxflow += e.f;

    vector<int> cut;
    for (int i = 0; i < m; i++) {
        auto &e = edges[i];
        if (dinic.vis[e.v] != dinic.vis[e.u])
            cut.push_back(i);
    }
}

```

2.7 KCC

```

void dfs1(int v, vector<char> &used, vector<int> &
    topsort) {
    used[v] = 1;
    for (auto u : g[v]) {
        if (!used[u])
            dfs1(u, used, topsort);
    }
    topsort.push_back(v);
}

void dfs2(int v, int col, vector<int> &comp) {
    comp[v] = col;
    for (auto u : rg[v]) {
        if (comp[u] == -1)
            dfs2(u, col, comp);
    }
}

signed main() {

```

```

    vector<int> topsort;
    topsort.reserve(n);
    vector<char> used(n, 0);
    for (int v = 0; v < n; ++v) {
        if (!used[v])
            dfs1(v, used, topsort);
    }
    reverse(all(topsort));
    int cc = 0;
    vector<int> comp(n, -1);
    for (int i = 0; i < n; ++i) {
        if (comp[topsort[i]] == -1)
            dfs2(topsort[i], cc++, comp);
    }
}

```

2.8 Кактус

```

// кратные рёбра можно, петли нельзя
void cactus_dfs(int v, int pei, vector<vector<int>> &
    c2vs, vector<int> &e2c, vector<vector<pair<int, int>>
    &g, vector<char> &used, vector<pair<int, int>>
    &st) {
    st.pb({v, pei});
    used[v] = 1;
    for (auto [u, ei] : g[v]) {
        if (ei == pei || e2c[ei] != -1)
            continue;
        if (used[u]) {
            int c = c2vs.size();
            c2vs.emplace_back();
            for (int j = st.size() - 1; st[j].first != u; --j)
            {
                c2vs[c].pb(st[j].first);
                e2c[st[j].second] = c;
            }
            c2vs[c].pb(u);
            e2c[ei] = c;
            continue;
        }
        cactus_dfs(u, ei, c2vs, e2c, g, used, st);
    }
    st.pop_back();
}

signed main() {
    // g[v][i] = {u, ei}, граф
    // e2c[ei] = номер цикла ребра ei (либо -1)
    // c2vs[c] = список вершин цикла c (в порядке обхода)
    vector<int> e2c(m, -1);
    vector<vector<int>> c2vs;
    c2vs.reserve(m - n + 1); // кол-во циклов = m-n+1
    {
        vector<char> used(n, 0);
        vector<pair<int, int>> st;
        st.reserve(n);
        for (int v = 0; v < n; ++v) {
            if (!used[v])
                cactus_dfs(v, -1, c2vs, e2c, g, used, st);

```

```

    }
}
// создание дерева по кактусу
vector<vector<int>> ngraph(n + c2vs.size());
for (int v = 0; v < n; ++v) {
    for (auto [u, ei] : g[v]) {
        if (e2c[ei] == -1)
            ngraph[v].pb(u);
    }
}
for (int c = 0; c < c2vs.size(); ++c) {
    for (int v : c2vs[c]) {
        ngraph[n + c].pb(v);
        ngraph[v].pb(n + c);
    }
}
}

```

2.9 Минкост (Джонсон)

```

using cost_t = ll;
using flow_t = int;

const int MAXN = 10000;
const int MAXM = 25000 * 2;
const cost_t INFw = 1e12;
const flow_t INFf = 10;

struct Edge {
    int v, u;
    flow_t f, c;
    cost_t w;
};

Edge edg[MAXN];
int esz = 0;
vector<int> graph[MAXN];
ll dist[MAXN];
ll pot[MAXN];
int S, T;
int NUMV;
int pre[MAXN];
bitset<MAXN> inQ;

flow_t get_flow() {
    int v = T;
    if (pre[v] == -1)
        return 0;
    flow_t f = INFf;
    do {
        int ei = pre[v];
        Edge &e = edg[ei];
        f = min(f, e.c - e.f);
        if (f == 0)
            return 0;
        v = e.v;
    } while (v != S);
    v = T;

```

```

do {
    int ei = pre[v];
    edg[ei].f += f;
    edg[ei ^ 1].f -= f;
    v = edg[ei].v;
} while (v != S);
return f;
}

void spfa() {
    fill(dist, dist + NUMV, INFw);
    dist[S] = 0;
    deque<int> Q = {S};
    inQ[S] = true;
    while (!Q.empty()) {
        int v = Q.front();
        Q.pop_front();
        inQ[v] = false;
        cost_t d = dist[v];
        for (int ei : graph[v]) {
            Edge &e = edg[ei];
            if (e.f == e.c)
                continue;
            cost_t w = e.w + pot[v] - pot[e.u];
            if (dist[e.u] <= d + w)
                continue;
            pre[e.u] = ei;
            dist[e.u] = d + w;
            if (!inQ[e.u]) {
                inQ[e.u] = true;
                Q.push_back(e.u);
            }
        }
        for (int i = 0; i < NUMV; ++i)
            pot[i] += dist[i];
    }

    cost_t mincost() {
        spfa(); // pot[i] = 0 // or ford_bellman
        flow_t f = 0;
        while (true) {
            flow_t ff = get_flow();
            if (ff == 0)
                break;
            f += ff;
            spfa(); // or dijkstra
        }
        cost_t res = 0;
        for (int i = 0; i < esz; ++i)
            res += edg[i].f * edg[i].w;
        res /= 2;
        return res;
    }

    void add_edge(int v, int u, int c, int w) {
        edg[esz] = {v, u, 0, c, w};
        edg[esz + 1] = {u, v, 0, 0, -w};
        graph[v].push_back(esz);
        graph[u].push_back(esz + 1);
    }
}

```

```

    esz += 2;
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    S = 0;
    T = n - 1;
    NUMV = n;
    for (int i = 0; i < m; ++i) {
        int v, u, c, w;
        cin >> v >> u >> c >> w;
        v--, u--;
        add_edge(v, u, c, w);
    }
    cost_t ans = mincost();
    cout << ans;
}

```

2.10 Мосты

```

// graph[v][i] = {u, edge_i}
void dfs(int v, int pi = -1) {
    vis[v] = 1;
    up[v] = tin[v] = timer++;
    for (auto [u, ei] : g[v]) {
        if (!vis[u]) {
            dfs(u, ei);
            up[v] = min(up[v], up[u]);
        } else if (ei != pi)
            up[v] = min(up[v], tin[u]);
        if (up[u] > tin[v]) {
            bridges.emplace_back(v, u);
            is_bridge[ei] = 1;
        }
    }
}

```

2.11 Паросочетания

```

bool dfs(int v, int c) {
    if (used[v] == c) return 0;
    used[v] = c;
    for (auto u : g[v]) {
        if (res[u] == -1) {
            res[u] = v;
            return 1;
        }
    }
    for (auto u : g[v]) {
        if (dfs(res[u], c)) {
            res[u] = v;
            return 1;
        }
    }
}

```

```

return 0;
}

signed main() {
    // n - в левой доле, m - в правой
    fill(res, res + m, -1);
    int ans = 0;
    for (int i = 0; i < n; ++i) {
        ans += dfs(i, ans + 1);
    }
}

```

2.12 Точки сочленения

```

void dfs(int v, int p = -1) {
    vis[v] = 1;
    up[v] = tin[v] = timer++;
    bool is_artic = false;
    int child = 0;
    for (auto u : g[v]) {
        if (!vis[u]) {
            child++;
            dfs(u, v);
            up[v] = min(up[v], up[u]);
            is_artic |= p != -1 && up[u] >= tin[v];
        } else
            up[v] = min(up[v], tin[u]);
    }
    is_artic |= p == -1 && child >= 2;
    if (is_artic)
        articulation_points.insert(v);
}

```

2.13 Эдмондс-Карп

```

struct edge {
    int v, f, c, ind;
};

vector<edge> g[MAXN];

bool bfs(int start, int final, int W) {
    vector<int> d(MAXN, INF);
    vector<pair<int, int>> pred(MAXN);
    d[start] = 0;
    deque<int> q = {start};
    while (!q.empty()) {
        int v = q.front();
        q.pop_front();
        for (int i = 0; i < (int) g[v].size(); i++) {
            auto e = g[v][i];
            if (e.f + W <= e.c && d[e.v] > d[v] + 1) {
                d[e.v] = d[v] + 1;
                pred[e.v] = {v, i};
                q.push_back(e.v);
            }
        }
    }
}

```

```

}
if (d[final] == INF) {
    return false;
}
int v = final;
int x = INF;
while (v != start) {
    int ind = pred[v].second;
    v = pred[v].first;
    x = min(x, g[v][ind].c - g[v][ind].f);
}
v = final;
while (v != start) {
    int ind = pred[v].second;
    v = pred[v].first;
    g[v][ind].f += x;
    g[g[v][ind].v][g[v][ind].ind].f -= x;
}
return true;
}

signed main() {
    int n, m;
    for (int i = 0; i < m; i++) {
        int u, v, c;
        cin >> u >> v >> c;
        g[u].push_back({v, 0, c, (int) g[v].size()});
        g[v].push_back({u, 0, 0, (int) g[u].size() - 1});
    }
    int start = 1, final = n;
    int W = (1 << 30);
    do {
        while (bfs(start, final, W));
        W /= 2;
    } while (W >= 1);
    int res = 0;
    for (auto e : g[start]) {
        res += e.f;
    }
}

```

2.14 Эйлеров цикл

// Эйлеров путь/цикл в компоненте связности s. Возвращает
индексы рёбер. Если пути/цикла нет, то алгоритм найдёт
фигню.

// Если неориентированный граф, то edges[ei] и edges[ei
~ 1] - обратные друг к другу рёбра.

// edges[graph[v][i]] = {v, u}

```

vector<int> eulerpath1(int s, vector<vector<int>> &graph,
    vector<pair<int, int>> &edges, vector<char> &used,
    vector<int> &start) {
    vector<pair<int, int>> st = {{-1, s}};
    vector<int> res;
    while (!st.empty()) {
        auto [ei, v] = st.back();
        while (start[v] < graph[v].size() && used[graph[v][start[v]]])
            start[v]++;
    }
}

```

```

if (start[v] == graph[v].size()) {
    if (ei != -1) res.push_back(ei);
    st.pop_back();
} else {
    int ej = graph[v][start[v]++];
    used[ej] = true;
    used[ej ^ 1] = true; // Удалить если ориент. граф
    st.emplace_back(ej, edges[ej].second);
}
}
reverse(all(res));
return res;
}

vector<char> used(edges.size(), false);
vector<int> start(graph.size(), 0);
for (int v = 0; v < graph.size(); ++v) {
    // Если ориентированный граф, второе условие заменить
    // на cnt_in[v] >= cnt_out[v]
    if (start[v] == graph[v].size() || graph[v].size() % 2 == 0)
        continue;
    auto path = eulerpath1(v, graph, edges, used, start);
}
for (int v = 0; v < graph.size(); ++v) {
    if (start[v] == graph[v].size())
        continue;
    auto cycle = eulerpath1(v, graph, edges, used, start);
}

```

3 ДП

3.1 ЧНТ

```

struct line {
    int k, b;
    int eval(int x) {
        return k * x + b;
    }
};

struct part {
    line a;
    double x;
};

double intersection(line a, line b) {
    return (a.b - b.b) / (double) (b.k - a.k);
}

struct ConvexHull {
    // for min: k decreasing (non-increasing)
    // for max: k increasing (non-decreasing)
    vector<part> st;

    void add(line a) {
        if (!st.empty() && st.back().a.k == a.k) {
            if (st.back().a.b < a.b) st.pop_back(); // for
            max
        }
        // if (st.back().a.b > a.b) st.pop_back(); // for
        min
        else return;
    }
}

```

```

}
while (st.size() > 1 &&
    intersection(st[st.size() - 2].a, a) <= st[
st.size() - 2].x)
    st.pop_back();
if (!st.empty()) st.back().x = intersection(st.
back().a, a);
st.push_back({a, INFINITY}); // C++ define
}

int get_val(int x) {
    if (st.empty()) {
        return -INF; // min possible value, for max
        return INF; // max possible value, for min
    }
    int l = -1, r = (int) st.size() - 1;
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (st[m].x < x) l = m;
        else r = m;
    }
    return st[r].a.eval(x);
}
};

```

3.2 Li Chao

```

// MAXIMUM
struct Line {
    int k, b;

    int f(int x) {
        return k * x + b;
    }
};

struct ST {
    vector<Line> st;

    ST(int n) {
        Line ln = {0LL, -INF};
        st.resize(4 * n, ln);
    }

    void upd(int i, int l, int r, Line ln) {
        int child = 1;
        Line ln1 = ln;
        int m = (l + r) / 2;
        if (ln.f(m) > st[i].f(m)) {
            if (ln.k < st[i].k) {
                child = 2;
            }
            ln1 = st[i];
            st[i] = ln;
        } else {
            if (st[i].k < ln.k) {
                child = 2;
            }
        }
    }
}

```

```

    if (l + 1 < r) {
        if (child == 1) {
            upd(i * 2 + 1, l, m, ln1);
        } else {
            upd(i * 2 + 2, m, r, ln1);
        }
    }

    int res(int i, int l, int r, int x) {
        if (l + 1 == r) {
            return st[i].f(x);
        }
        int m = (l + r) / 2;
        int val = st[i].f(x);
        if (x < m) {
            val = max(val, res(i * 2 + 1, l, m, x));
        } else {
            val = max(val, res(i * 2 + 2, m, r, x));
        }
        return val;
    }
};

```

3.3 SOS-dp

```

// dp initial fill, a[] is given array, mb extra zeros
for (int i = 0; i < (1 << N); i++) {
    dp[i] = a[i];
}

// Classic SOS-dp, goal: dp[mask] = \sum a[submasks of mask]
for (int i = 0; i < N; i++) {
    for (int mask = 0; mask < (1 << N); mask++) {
        if ((mask >> i) & 1) {
            dp[mask] += dp[mask ^ (1 << i)];
        }
    }
}

// Overmasks SOS-dp, goal: dp[mask] = \sum a[overmasks of mask]
for (int i = 0; i < N; i++) {
    for (int mask = (1 << N) - 1; mask >= 0; mask--) {
        if (((mask >> i) & 1) == 0) {
            dp[mask] += dp[mask ^ (1 << i)];
        }
    }
}

// to inverse SOS-dp (restore original array by SOS-dp array):
// use same code, but -= instead of += in dp transitions

```

3.4 HBП

```

// 0-indexation ({a0, ..., an-1})
vector<int> lis(vector<int> a) {
    int n = (int) a.size();
    vector<int> dp(n + 1, INF), ind(n + 1), par(n + 1); // INF > all a[i] required
    ind[0] = -INF;
    dp[0] = -INF;
    for (int i = 0; i < n; i++) {
        int l = upper_bound(dp.begin(), dp.end(), a[i]) - dp.begin();
        if (dp[l - 1] < a[i] && a[i] < dp[l]) {
            dp[l] = a[i];
            ind[l] = i;
            par[i] = ind[l - 1];
        }
    }
    vector<int> ans; // exact values
    for (int l = n; l >= 0; l--) {
        if (dp[l] < INF) {
            int pi = ind[l];
            ans.resize(l);
            for (int i = 0; i < l; i++) {
                ans[i] = a[pi]; // =pi if need indices
                pi = par[pi];
            }
            reverse(ans.begin(), ans.end());
            return ans;
        }
    }
    return {};
}

```

3.5 HOПП

```

// 1-indexation ({0, a1, ..., an}, {0, b1, ..., bm})
vector<int> lcis(vector<int> a, vector<int> b) {
    int n = (int) a.size() - 1, m = (int) b.size() - 1;
    vector<int> dp(m + 1), dp2(m + 1), par(m + 1);
    for (int i = 1; i <= n; i++) {
        int best = 0, best_idx = 0;
        for (int j = 1; j <= m; j++) {
            dp2[j] = dp[j];
            if (a[i] == b[j]) {
                dp2[j] = max(dp2[j], best + 1);
                par[j] = best_idx;
            }
            if (a[i] > b[j] && best < dp[j]) {
                best = dp[j];
                best_idx = j;
            }
        }
        swap(dp, dp2);
    }
    int pj = 0;
    for (int j = 1; j <= m; j++) {
        if (dp[pj] < dp[j]) {
            pj = j;
        }
    }
}

```

```

vector<int> ans; // exact values
while (pj > 0) {
    ans.push_back(b[pj]);
    pj = par[pj];
}
reverse(ans.begin(), ans.end());
return ans;
}

```

4 Деревья

4.1 Centroid

```

int levels[MAXN];
int szs[MAXN];
int cent_par[MAXN];

int calcsizes(int v, int p) {
    int sz = 1;
    for (int u : graph[v]) {
        if (u != p && levels[u] == 0)
            sz += calcsizes(u, v);
    }
    return szs[v] = sz;
}

void centroid(int v, int lvl=1, int p=-1) {
    int sz = calcsizes(v, -1);
    int nxt = v, prv;
    while (nxt != -1) {
        prv = v, v = nxt, nxt = -1;
        for (int u : graph[v]) {
            if (u != prv && levels[u] == 0 && szs[u] * 2 >= sz)
                nxt = u;
        }
        levels[v] = lvl;
        cent_par[v] = p;
        for (int u : graph[v]) {
            if (levels[u] == 0)
                centroid(u, lvl + 1, v);
        }
        // calc smth for centroid v
    }
}

```

4.2 HLD

```

int par[MAXN], sizes[MAXN];
int pathup[MAXN];
int tin[MAXN], tout[MAXN];
int timer;

int dfs1_hld(int v, int p) {
    par[v] = p;
    int sz = 1;
    for (int i = 0; i < graph[v].size(); ++i) {

```

```

    int u = graph[v][i];
    if (u == p) {
        swap(graph[v][i--], graph[v].back());
        graph[v].pop_back();
        continue;
    }
    sz += dfs1_hld(u, v);
}
return sizes[v] = sz;
}

void dfs2_hld(int v, int up) {
    tin[v] = timer++;
    pathup[v] = up;
    if (graph[v].empty()) {
        tout[v] = timer;
        return;
    }
    for (int i = 1; i < graph[v].size(); ++i) {
        if (sizes[graph[v][i]] > sizes[graph[v][0]])
            swap(graph[v][i], graph[v][0]);
    }
    dfs2_hld(graph[v][0], up);
    for (int i = 1; i < graph[v].size(); ++i)
        dfs2_hld(graph[v][i], graph[v][0]);
    tout[v] = timer;
}

bool is_ancestor(int v, int p) {
    return tin[p] <= tin[v] && tout[v] <= tout[p];
}

// get_hld полностью аналогичный
void update_hld(int v, int u, int ARG) {
    for (int _ = 0; _ < 2; ++_) {
        while (!is_ancestor(u, pathup[v])) {
            int vup = pathup[v];
            ST.update(0, 0, timer, tin[vup], tin[v] + 1, ARG);
            v = par[vup];
        }
        swap(v, u);
    }
    if (tin[v] > tin[u])
        swap(v, u);
    // v = lca
    ST.update(0, 0, timer, tin[v], tin[u] + 1, ARG);
}

signed main() {
    dfs1_hld(0, -1);
    dfs2_hld(0, 0);
    ST.build();
    // your code here
}

```

4.3 Link-cut

```

struct Node {
    Node *ch[2];

```

```

    Node *p;
    bool rev;
    int sz;

    Node() {
        ch[0] = nullptr;
        ch[1] = nullptr;
        p = nullptr;
        rev = false;
        sz = 1;
    }
};

int size(Node *v) {
    return (v ? v->sz : 0);
}

int chnum(Node *v) {
    return v->p->ch[1] == v;
}

bool isroot(Node *v) {
    return v->p == nullptr || v->p->ch[chnum(v)] != v;
}

void push(Node *v) {
    if (v->rev) {
        if (v->ch[0])
            v->ch[0]->rev ^= 1;
        if (v->ch[1])
            v->ch[1]->rev ^= 1;
        swap(v->ch[0], v->ch[1]);
        v->rev = false;
    }
}

void pull(Node *v) {
    v->sz = size(v->ch[1]) + size(v->ch[0]) + 1;
}

void attach(Node *v, Node *p, int num) {
    if (p)
        p->ch[num] = v;
    if (v)
        v->p = p;
}

void rotate(Node *v) {
    Node *p = v->p;
    push(p);
    push(v);
    int num = chnum(v);
    Node *u = v->ch[1 - num];
    if (!isroot(v->p))
        attach(v, p->p, chnum(p));
    else
        v->p = p->p;
    attach(u, p, num);
    attach(p, v, 1 - num);
    pull(p);
}

```

```

    pull(v);
}

void splay(Node *v) {
    push(v);
    while (!isroot(v)) {
        if (!isroot(v->p)) {
            if (chnum(v) == chnum(v->p))
                rotate(v->p);
            else
                rotate(v);
        }
        rotate(v);
    }
}

void expose(Node *v) {
    splay(v);
    v->ch[1] = nullptr;
    pull(v);
    while (v->p != nullptr) {
        Node *p = v->p;
        splay(p);
        attach(v, p, 1);
        pull(p);
        splay(v);
    }
}

void makeroot(Node *v) {
    expose(v);
    v->rev ^= 1;
    push(v);
}

void link(Node *v, Node *u) {
    makeroot(v);
    makeroot(u);
    u->p = v;
}

void cut(Node *v, Node *u) {
    makeroot(u);
    makeroot(v);
    v->ch[1] = nullptr;
    u->p = nullptr;
}

int get(Node *v, Node *u) {
    makeroot(u);
    makeroot(v);
    Node *w = u;
    while (!isroot(w))
        w = w->p;
    return (w == v ? size(v) - 1 : -1);
}

const int MAXN = 100010;
Node *nodes[MAXN];

```



```
int main() {
    int n, q;
    cin >> n >> q;
    for (int i = 0; i < n; ++i)
        nodes[i] = new Node();
    while (q--) {
        string s;
        int a, b;
        cin >> s >> a >> b;
        a--, b--;
        if (s[0] == 'g')
            cout << get(nodes[a], nodes[b]) << '\n';
        else if (s[0] == 'l')
            link(nodes[a], nodes[b]);
        else
            cut(nodes[a], nodes[b]);
    }
}
```

5 Другое

5.1 Fast mod

```
// Быстрое взятие по НЕ константному модулю (в 2-4 раза
// быстрее)
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m(-1ULL / b) {}
    ull mod(ull a) const {
        ull r = a - (ull)((__uint128_t(m) * a) >> 64) * b;
        return r; // r in [0, 2b) // ≈ x3.5 speed
        return r >= b ? r - b : r; // ≈ x3 speed
    }
}; // Usage:
// FastMod F(m);
// ull x_mod_m = F.mod(x);
```

5.2 Slope trick

// Дан массив a_n . Сделать минимальное кол-во ± 1 , чтобы a_n стал неубывающим.

```
void solve() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    int ans = 0;
    multiset<int> now;
    for (int i = 0; i < n; i++) {
        now.insert(a[i]);
        ans += (*now.rbegin() - a[i]);
        now.erase(now.find(*now.rbegin()));
        now.insert(a[i]);
    }
}
```

```
cout << ans << '\n';
}
```

5.3 attribute_packed

```
struct Kek {
    int a;
    char b;
    // char[3]
    int c;
} __attribute__((packed));
// sizeof = 9 (instead of 12)
```

5.4 custom_bitset

```
// __builtin_ctz = Count Trailing Zeroes
// __builtin_clz = Count Leading Zeroes
// both are UB in gcc when pass 0
struct custom_bitset {
    vector<uint64_t> bits;
    int b, n;

    custom_bitset(int _b = 0) {
        init(_b);
    }
    void init(int _b) {
        b = _b, n = (b + 63) / 64;
        bits.assign(n, 0);
    }
    void clear() {
        b = n = 0;
        bits.clear();
    }
    void reset() {
        bits.assign(n, 0);
    }
    void _clean() {
        // Reset all bits after 'b'.
        if (b != 64 * n)
            bits.back() &= (1LLU << (b - 64 * (n - 1))) - 1;
    }
    bool get(int i) const {
        return bits[i / 64] >> (i % 64) & 1;
    }
    void set(int i, bool value) {
        // assert(0 <= i && i < b);
        bits[i / 64] &= ~(1LLU << (i % 64));
        bits[i / 64] |= uint64_t(value) << (i % 64);
    }

    // Simulates 'bs |= bs << shift;'
    // '|= ' can be replaced with '^=', '&=', '!='
    void or_shift_left(int shift) {
        int div = shift / 64, mod = shift % 64;
        if (mod == 0) {
            for (int i = n - 1; i >= div; i--)
                bits[i] |= bits[i - div];
        }
    }
}
```

```
} else {
    for (int i = n - 1; i >= div + 1; i--)
        bits[i] |= bits[i - (div + 1)] >> (64 - mod) |
        bits[i - div] << mod;
    if (div < n)
        bits[div] |= bits[0] << mod;
}
// if '&=', '^='
// fill(bits.begin(), bits.begin() + min(div, n), 0);
_clean();

// Simulates 'bs |= bs >> shift;'
// '|= ' can be replaced with '^=', '&=', '!='
void or_shift_right(int shift) {
    int div = shift / 64, mod = shift % 64;
    if (mod == 0) {
        for (int i = div; i < n; i++)
            bits[i - div] |= bits[i];
    } else {
        for (int i = 0; i < n - (div + 1); i++)
            bits[i] |= bits[i + (div + 1)] << (64 - mod) |
            bits[i + div] >> mod;
        if (div < n)
            bits[n - div - 1] |= bits[n - 1] >> mod;
    }
    // if '&=', '^='
    // fill(bits.end() - min(div, n), bits.end(), 0);
    _clean();

    // find min j, that j >= i and bs[j] = 1;
    int find_next(int i) {
        if (i >= b) return b;
        int div = i / 64, mod = i % 64;
        auto x = bits[div] >> mod;
        if (x != 0)
            return i + __builtin_ctzll(x);
        for (auto k = div + 1; k < n; ++k) {
            if (bits[k] != 0)
                return 64 * k + __builtin_ctzll(bits[k]);
        }
        return b;
    }

    // '|= ' can be replaced with '&=', '^='
    custom_bitset &operator|=(const custom_bitset &other) {
        // assert(b == other.b);
        for (int i = 0; i < n; i++)
            bits[i] |= other.bits[i];
        return *this;
    }
};
```

5.5 ordered_set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
```

```
using namespace __gnu_pbds;

typedef tree<int, null_type, less<>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;

//st.find_by_order(index);
//st.order_of_key(key);
```

5.6 pragma

```
#pragma GCC optimize("Ofast,fast-math,unroll-loops,no-
    stack-protector,inline")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.1,sse4
    .2,avx,avx2,abm,mmx,popcnt")
```

5.7 Аллокатор Копелиовича

// Код вставить до инклюдов

```
#include <cassert>
```

```
const int MAX_MEM = 1e8; // ~100mb
int mpos = 0;
char mem[MAX_MEM];
```

```
inline void *operator new(std::size_t n) {
    mpos += n;
    // assert(mpos <= MAX_MEM);
    return (void *) (mem + mpos - n);
}
```

```
inline void operator delete(void *) noexcept {} // must
    have!
```

```
inline void operator delete(void *, std::size_t)
    noexcept {} // fix!!
```

5.8 Альфа-бета отсечение

```
int alphabeta(int player, int alpha, int beta, int depth) {
    if (depth == 0) {
        // return current position score
    }
    if (player == 0) { // maximization player
        int val = -INF;
        for (auto move : possible_moves) {
            val = max(val, alphabeta(1, alpha, beta, depth -
                1));
            if (val > beta) break;
            alpha = max(alpha, val);
        }
        return val;
    } else {
        int val = INF;
        for (auto move : possible_moves) {
```

```
        val = min(val, alphabeta(0, alpha, beta, depth -
            1));
        if (val < alpha) break;
        beta = min(beta, val);
    }
    return val;
}
```

5.9 Отжиг

```
const double lambda = 0.999;
double temprature = 1;
mt19937 rnd(777);

double gen_rand_01() {
    return rnd() / (double) UINT32_MAX;
}

bool f(int delta) {
    return exp(-delta / temprature) > gen_rand_01();
}

void make_change() {
    temprature *= lambda;
    // calc change score
    if (change_score <= 0 || f(change_score)) {
        score += change_score;
        // make change
    }
}
```

6 Математика

6.1 AdivB cmp CdivD

```
char sign(ll x) {
    return x < 0 ? -1 : x > 0;
}

// -1 = less, 0 = equal, 1 = greater
char compare(ll a, ll b, ll c, ll d) {
    if (a / b != c / d)
        return sign(a / b - c / d);
    a = a % b;
    c = c % d;
    if (a == 0)
        return -sign(c) * sign(d);
    if (c == 0)
        return sign(a) * sign(b);
    return compare(d, c, b, a) * sign(a) * sign(b) * sign(
        c) * sign(d);
}
```

6.2 Berlecamp

```
int getkfps(vector<ll> p, vector<ll> q, ll k) {
```

```
// assert(q[0] != 0);
while (k) {
    auto f = q;
    for (int i = 1; i < (int) f.size(); i += 2) {
        f[i] = (MOD - f[i] % MOD) % MOD;
    }
    auto p2 = convMod(p, f);
    auto q2 = convMod(q, f);
    p.clear(), q.clear();
    for (int i = k % 2; i < (int) p2.size(); i += 2) {
        p.pb(p2[i]);
    }
    for (int i = 0; i < (int) q2.size(); i += 2) {
        q.pb(q2[i]);
    }
    k >>= 1;
}
return (int) ((p[0] * inverse(q[0])) % MOD);
}

// a - initials values of sequence, s - result of
// berlekamp on a
int kth_term(vector<ll> &a, vector<ll> s, ll k) {
    int d = ssize(s) - 1;
    s[0] = MOD - 1;
    while (s.back() == 0) {
        s.pop_back();
    }
    for (auto &el : s) {
        el = (MOD - el % MOD) % MOD;
    }
    vector<ll> p(d);
    copy(a.begin(), a.begin() + d, p.begin());
    p = convMod(p, s);
    p.resize(d);
    return getkfps(p, s, k);
}
```

```
vector<ll> berlekamp_massey(vector<ll> a) {
    // given a[0]...a[n], returns sequence s[1]..s[k] s.t
    // a[i] = a[i-1] \cdot s[1] + \ldots + a[i-k] \cdot s[
        k]
    vector<ll> ls, s;
    int lf = 0, d = 0;
    for (int i = 0; i < a.size(); ++i) {
        ll t = 0;
        for (int j = 0; j < s.size(); ++j) {
            t = (t + 1ll * a[i - j - 1] * s[j]) % MOD;
        }
        if ((t - a[i]) % MOD == 0) continue;
        if (s.empty()) {
            s.resize(i + 1);
            lf = i;
            d = (t - a[i]) % MOD;
            continue;
        }
        ll k = -(a[i] - t) * inverse(d) % MOD;
        vector<ll> c(i - lf - 1);
        c.push_back(k);
        for (auto &j : ls)
```

```

    c.push_back(-j * k % MOD);
    if (c.size() < s.size())
        c.resize(s.size());
    for (int j = 0; j < s.size(); ++j) {
        c[j] = (c[j] + s[j]) % MOD;
    }
    if (i - lf + (int) ls.size() >= (int) s.size()) {
        tie(ls, lf, d) = make_tuple(s, i, (t - a[i]) % MOD);
    }
    s = c;
}
s.insert(s.begin(), 0); // fictive s[0] = 0
for (auto &i: s)
    i = (i % MOD + MOD) % MOD;
return s;
}

```

6.3 FFT mod

```

const int MOD = 998244353; //  $7 \cdot 17 \cdot 2^{23} + 1$ 
const int G = 3;
//const int MOD = 7340033; //  $7 \cdot 2^{20} + 1$ 
//const int G = 5;
//const int MOD = 469762049; //  $7 \cdot 2^{26} + 1$ 
//const int G = 30;
const int MAXLOG = 23;
int W[(1 << MAXLOG) + 10];
bool nttinit = false;
vector<int> pws;

// int add(), int sub(), int mul(),
// int binpow(), int inv()

```

```

void initNTT() {
    if (nttinit) return;
    nttinit = true;
    assert((MOD - 1) % (1 << MAXLOG) == 0);
    pws.push_back(binpow(G, (MOD - 1) / (1 << MAXLOG)));
    for (int i = 0; i < MAXLOG - 1; ++i)
        pws.push_back(mul(pws.back(), pws.back()));
    assert(pws.back() == MOD - 1);
    W[0] = 1;
    for (int i = 1; i < (1 << MAXLOG); ++i)
        W[i] = mul(W[i - 1], pws[0]);
}

```

```

void ntt(int n, vector<int> &a, bool rev) {
    initNTT();
    int lg = 31 - __builtin_clz(n);
    for (auto &el : a) {
        el = (el % MOD + MOD) % MOD;
    }
    vector<int> rv(n);
    for (int i = 1; i < n; ++i) {
        rv[i] = (rv[i >> 1] >> 1) ^ ((i & 1) << (lg - 1));
        if (rv[i] > i) swap(a[i], a[rv[i]]);
    }
}

```

```

int num = MAXLOG - 1;
for (int len = 1; len < n; len *= 2, --num) {
    for (int i = 0; i < n; i += 2 * len) {
        for (int j = 0; j < len; ++j) {
            int u = a[i + j];
            int v = mul(W[j << num], a[i + j + len]);
            a[i + j] = add(u, v);
            a[i + j + len] = sub(u, v);
        }
    }
}
if (rev) {
    int invn = binpow(n, MOD - 2);
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], invn);
    reverse(a.begin() + 1, a.end());
}
}

```

```

vector<int> conv(vector<int> a, vector<int> b) {
    if (a.empty() || b.empty())
        return {};
    int lg = 32 - __builtin_clz(a.size() + b.size() - 1);
    int n = 1 << lg;
    a.resize(n);
    b.resize(n);
    ntt(n, a, false);
    ntt(n, b, false);
    for (int i = 0; i < n; ++i)
        a[i] = mul(a[i], b[i]);
    ntt(n, a, true);
    while (a.size() > 1 && a.back() == 0)
        a.pop_back();
    return a;
}

```

```

vector<int> add(vector<int> a, vector<int> b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < (int) b.size(); ++i)
        a[i] = add(a[i], b[i]);
    return a;
}

```

```

vector<int> sub(vector<int> a, vector<int> b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < (int) b.size(); ++i)
        a[i] = sub(a[i], b[i]);
    return a;
}

```

```

vector<int> inv(const vector<int> &a, int need) {
    vector<int> b = {inv(a[0])};
    while ((int) b.size() < need) {
        vector<int> a1 = a;
        int m = b.size();
        a1.resize(min((int) a1.size(), 2 * m));
        b = conv(b, sub({2}, conv(a1, b)));
        b.resize(2 * m);
    }
    b.resize(need);
    return b;
}

```

```

}

vector<int> div(vector<int> a, vector<int> b) {
    if (count(all(a), 0) == a.size())
        return {0};
    assert(a.back() != 0 && b.back() != 0);
    int n = a.size() - 1;
    int m = b.size() - 1;
    if (n < m)
        return {0};
    reverse(all(a));
    reverse(all(b));
    a.resize(n - m + 1);
    b.resize(n - m + 1);
    vector<int> c = inv(b, b.size());
    vector<int> q = conv(a, c);
    q.resize(n - m + 1);
    reverse(all(q));
    return q;
}

```

```

vector<int> mod(vector<int> a, vector<int> b) {
    auto res = sub(a, conv(b, div(a, b)));
    while (res.size() > 1 && res.back() == 0)
        res.pop_back();
    return res;
}

```

```

vector<int> multipoint(vector<int> a, vector<int> x) {
    int n = x.size();
    vector<vector<int>> tree(2 * n);
    for (int i = 0; i < n; ++i)
        tree[i + n] = {x[i], MOD - 1};
    for (int i = n - 1; i >= 0; --i)
        tree[i] = conv(tree[2 * i], tree[2 * i + 1]);
    tree[1] = mod(a, tree[1]);
    for (int i = 2; i < 2 * n; ++i)
        tree[i] = mod(tree[i >> 1], tree[i]);
    vector<int> res(n);
    for (int i = 0; i < n; ++i)
        res[i] = tree[i + n][0];
    return res;
}

```

```

vector<int> deriv(vector<int> a) {
    for (int i = 1; i < (int) a.size(); ++i)
        a[i - 1] = mul(i, a[i]);
    a.back() = 0;
    if (a.size() > 1)
        a.pop_back();
    return a;
}

```

```

vector<int> integ(vector<int> a) {
    a.push_back(0);
    for (int i = (int) a.size() - 1; i >= 0; --i)
        a[i] = mul(a[i - 1], inv(i));
    a[0] = 0;
    return a;
}

```

```
vector<int> log(vector<int> a, int n) {
    assert(a[0] == 1);
    auto res = integ(conv(deriv(a), inv(a, n)));
    res.resize(n);
    return res;
}

vector<int> exp(vector<int> a, int need) {
    assert(a[0] == 0);
    vector<int> b = {1};
    while ((int) b.size() < need) {
        vector<int> a1 = a;
        int m = b.size();
        a1.resize(min((int) a1.size(), 2 * m));
        a1[0] = add(a1[0], 1);
        b = conv(b, sub(a1, log(b, 2 * m)));
        b.resize(2 * m);
    }
    b.resize(need);
    return b;
}
```

6.4 FFT

```
const double PI = acos(-1);
const int LOG = 20;
const int MAXN = 1 << LOG;
```

```
//using comp = complex<double>;
struct comp {
    double x, y;
    comp() : x(0), y(0) {}
    comp(double x, double y) : x(x), y(y) {}
    comp(int x) : x(x), y(0) {}
    comp operator+(const comp &o) const { return {x + o.x,
        y + o.y}; }
    comp operator-(const comp &o) const { return {x - o.x,
        y - o.y}; }
    comp operator*(const comp &o) const { return {x * o.x
        - y * o.y, x * o.y + y * o.x}; }
    comp operator/(const int k) const { return {x / k, y /
        k}; }
    comp conj() const { return {x, -y}; }
};

comp OMEGA[MAXN + 10];
int tail[MAXN + 10];

comp omega(int n, int k) {
    return OMEGA[MAXN / n * k];
}

int gettail(int x, int lg) {
    return tail[x] >> (LOG - lg);
}

void calcomega() {
    for (int i = 0; i < MAXN; ++i) {
```

```
        double x = 2 * PI * i / MAXN;
        OMEGA[i] = {cos(x), sin(x)};
    }
}

void calctail() {
    tail[0] = 0;
    for (int i = 1; i < MAXN; ++i)
        tail[i] = (tail[i >> 1] >> 1) | ((i & 1) << (LOG -
            1));
}

void fft(vector<comp> &A, int lg) {
    int n = A.size();
    for (int i = 0; i < n; ++i) {
        int j = gettail(i, lg);
        if (i < j)
            swap(A[i], A[j]);
    }
    for (int len = 2; len <= n; len *= 2) {
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < len / 2; ++j) {
                auto v = A[i + j];
                auto u = A[i + j + len / 2] * omega(len, j);
                A[i + j] = v + u;
                A[i + j + len / 2] = v - u;
            }
        }
    }
}

void fft2(vector<comp> &A, vector<comp> &B, int lg) {
    int n = A.size();
    vector<comp> C(n);
    for (int i = 0; i < n; ++i) {
        C[i].x = A[i].x;
        C[i].y = B[i].x;
    }
    fft(C, lg);
    C.push_back(C[0]);
    for (int i = 0; i < n; ++i) {
        A[i] = (C[i] + C[n - i].conj()) / 2;
        B[i] = (C[i] - C[n - i].conj()) / 2 * comp(0, -1);
    }
}

void invfft(vector<comp> &A, int lg) {
    int n = 1 << lg;
    fft(A, lg);
    for (auto &el : A)
        el = el / n;
    reverse(A.begin() + 1, A.end());
}

vector<int> mul(vector<int> &a, vector<int> &b) {
    if (a.empty() || b.empty())
        return {};
    int lg = 32 - __builtin_clz(a.size() + b.size() - 1);
    int n = 1 << lg;
    vector<comp> A(n, 0), B(n, 0);
```

```
    for (int i = 0; i < a.size(); ++i)
        A[i] = a[i];
    for (int i = 0; i < b.size(); ++i)
        B[i] = b[i];
    // fft2(A, B, lg);
    fft(A, lg);
    fft(B, lg);
    for (int i = 0; i < n; ++i)
        A[i] = A[i] * B[i];
    invfft(A, lg);
    vector<int> c(n);
    for (int i = 0; i < n; ++i)
        c[i] = round(A[i].x);
    while (!c.empty() && c.back() == 0)
        c.pop_back();
    return c;
}

signed main() {
    calcomega(); // НЕ ЗАБЫТЬ
    calctail(); // НЕ ЗАБЫТЬ
    // your code here
}
```

6.5 Floor Sum

```
int floor_sum(int n, int d, int m, int a) {
    // sum_{i=0}^{n-1} floor((a + i*m)/d), only non-
    // negative integers!
    int ans = 0;
    ans += (n * (n - 1) / 2) * (m / d);
    m %= d;
    ans += n * (a / d);
    a %= d;
    int l = m * n + a;
    if (l >= d)
        ans += floor_sum(l / d, m, d, l % d);
    return ans;
}
```

6.6 GCD LCM свёртки

```
vector<int> gcd_convolution(vector<int> a, vector<int> b) {
    // a,b is 1-indexed array; a[0],b[0] doesnt matter
    int n = ssize(a) - 1;
    for (int p = 2; p <= n; ++p) {
        if (!isprime[p]) continue; // alt: for (auto p :
        primes) if (p > n) break;
        for (int i = n / p; i > 0; --i) {
            a[i] += a[i * p];
            if (a[i] >= mod) a[i] -= mod;
            b[i] += b[i * p];
            if (b[i] >= mod) b[i] -= mod;
        }
    }
}
```

```

    for (int i = 1; i <= n; ++i) a[i] = (a[i] * b[i]) %
        mod;
    for (int p = 2; p <= n; ++p) {
        if (!isprime[p]) continue;
        for (int i = 1; i * p <= n; ++i) {
            a[i] += mod - a[i * p];
            if (a[i] >= mod) a[i] -= mod;
        }
    }
    return a;
}

vector<int> lcm_convolution(vector<int> a, vector<int> b)
{
    int n = ssize(a) - 1;
    for (int p = 2; p <= n; ++p) {
        if (!isprime[p]) continue;
        for (int i = 1; i * p <= n; ++i) {
            a[i * p] += a[i];
            if (a[i * p] >= mod) a[i * p] -= mod;
            b[i * p] += b[i];
            if (b[i * p] >= mod) b[i * p] -= mod;
        }
    }
    for (int i = 1; i <= n; ++i) a[i] = (a[i] * b[i]) %
        mod;
    for (int p = 2; p <= n; ++p) {
        if (!isprime[p]) continue;
        for (int i = n / p; i > 0; --i) {
            a[i * p] += mod - a[i];
            if (a[i * p] >= mod) a[i * p] -= mod;
        }
    }
    return a;
}

```

6.7 Interpolation

```

struct poly {
    int n = 0;
    vector<ll> a = {0};
    poly() = default;
    poly(vector<ll> a) : a(a), n((int)a.size()-1) {}
    poly(int n) : n(n), a(n+1, 0) {}

    ll evaluate(int x) {
        ll val = 0, y = 1;
        for (int i = 0; i <= n; ++i) {
            val = (val + a[i] * y) % MOD;
            y = (y * x) % MOD;
        }
        return val;
    }
};

poly mul(poly &P, poly &Q) {
    poly R(P.n + Q.n);
    for (int i = 0; i <= P.n; ++i) {
        for (int j = 0; j <= Q.n; ++j) {

```

```

            R.a[i + j] = (R.a[i + j] + P.a[i] * Q.a[j]) % MOD;
        }
    }
    return R;
}

poly multiply_many(vector<poly> a) {
    if (a.empty()) return poly((vector<ll>){1ll});
    if (a.size() == 1) return a[0];
    vector<poly> b;
    int n = a.size();
    for (int i = n / 2; i < n; i++) {
        b.push_back(a[i]);
    }
    a.resize(n / 2);
    poly P = multiply_many(a), Q = multiply_many(b);
    return mul(P, Q);
}

poly divide(poly &res, int c) {
    poly ans(res.n - 1);
    ll val = 0;
    for (int i = res.n; i >= 1; i--) {
        val = (val * c + res.a[i]) % MOD;
        ans.a[i - 1] = val;
    }
    return ans;
}

poly interpolate(int deg, vector<int> x, vector<int> y)
{
    assert(x.size() >= deg + 1 && y.size() >= deg + 1);
    x.resize(deg + 1), y.resize(deg + 1);
    vector<poly> fm;
    for (int j = 0; j <= deg; ++j) {
        fm.push_back(poly({(MOD - x[j]) % MOD, 1}));
    }
    poly res = multiply_many(fm);
    poly ans(deg);
    for (int i = 0; i <= deg; ++i) {
        ll denom = 1;
        for (int j = 0; j <= deg; ++j) {
            if (j != i) {
                denom = denom * ((x[i] - x[j] + MOD) % MOD) %
                    MOD;
            }
        }
        poly res_divided = divide(res, x[i]);
        denom = inv(denom);
        denom = denom * y[i] % MOD;
        for (int j = 0; j <= deg; ++j) {
            ll el = res_divided.a[j];
            el = (el * denom) % MOD;
            ans.a[j] = (ans.a[j] + el) % MOD;
        }
    }
    return ans;
}

```

6.8 OR XOR AND свёртки

```

vector<int> or_conv(int n, vector<int> a, vector<int> b)
{
    // |a| = |b| = 2^n
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < (1 << n); ++j) {
            if ((j >> i) & 1) {
                a[j] = (a[j] + a[j ^ (1 << i)]) % MOD;
                b[j] = (b[j] + b[j ^ (1 << i)]) % MOD;
            }
        }
    }
    vector<int> c(1 << n);
    for (int i = 0; i < (1 << n); ++i) {
        c[i] = (a[i] * b[i]) % MOD;
    }
    for (int i = n - 1; i >= 0; i--) {
        for (int j = (1 << n) - 1; j >= 0; j--) {
            if ((j >> i) & 1) {
                c[j] = (c[j] - c[j ^ (1 << i)] + MOD) % MOD;
            }
        }
    }
    return c;
}

vector<int> and_conv(int n, vector<int> a, vector<int> b)
{
    // |a| = |b| = 2^n
    for (int i = 1; i < (1 << n); i *= 2) {
        for (int j = 0; j < (1 << n); j += i * 2) {
            for (int k = 0; k < i; k++) {
                a[j + k] = (a[j + k] + a[i + j + k]) % MOD;
                b[j + k] = (b[j + k] + b[i + j + k]) % MOD;
            }
        }
    }
    vector<int> c(1 << n);
    for (int i = 0; i < (1 << n); ++i) {
        c[i] = (a[i] * b[i]) % MOD;
    }
    for (int i = 1; i < (1 << n); i *= 2) {
        for (int j = 0; j < (1 << n); j += i * 2) {
            for (int k = 0; k < i; k++) {
                c[j + k] = (c[j + k] - c[i + j + k] + MOD) % MOD;
            }
        }
    }
    return c;
}

const int inv2 = (MOD + 1) / 2;
vector<int> xor_conv(int n, vector<int> a, vector<int> b)
{
    // |a| = |b| = 2^n
    for (int i = 1; i < (1 << n); i *= 2) {
        for (int j = 0; j < (1 << n); ++j) {
            if ((j & i) == 0) {
                int x = a[j], y = a[j | i];
                a[j] = (x + y) % MOD, a[j | i] = (x - y + MOD) %
                    MOD;
                x = b[j], y = b[j | i];

```



```

    b[j] = (x + y) % MOD, b[j | i] = (x - y + MOD) %
MOD;
}
}
vector<int> c(1 << n);
for (int i = 0; i < (1 << n); i++)
    c[i] = (a[i] * b[i]) % MOD;
for (int i = 1; i < (1 << n); i *= 2) {
    for (int j = 0; j < (1 << n); j++) {
        if ((j & i) == 0) {
            int x = c[j], y = c[j | i];
            c[j] = (inv2 * (x + y)) % MOD, c[j | i] = (inv2
* (x - y + MOD)) % MOD;
        }
    }
}
return c;
}

```

6.9 convMod

```

vector<ll> convMod(const vector<ll> &a, const vector<ll>
&b) {
    if (a.empty() || b.empty()) return {};
    vector<ll> res((int) a.size() + (int) b.size() - 1);
    int lg = 32 - __builtin_clz((int) res.size()), n = 1
<< lg, cut = (int) (sqrt(MOD));
    vector<comp> L(n), R(n), outs(n), outl(n);
    for (int i = 0; i < a.size(); i++) L[i] = comp((int) a
[i] / cut, (int) a[i] % cut);
    for (int i = 0; i < b.size(); i++) R[i] = comp((int) b
[i] / cut, (int) b[i] % cut);
    fft(L, lg), fft(R, lg);
    for (int i = 0; i < n; i++) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + L[j].conj()) * R[i] / (2.0 * n);
        outs[j] = (L[i] - L[j].conj()) * R[i] / (2.0 * n) *
comp(0, 1) * -1;
    }
    fft(outl, lg), fft(outs, lg);
    for (int i = 0; i < res.size(); i++) {
        ll av = (ll)((outl[i]).x + .5), cv = (ll)((outs[i]).
y + .5);
        ll bv = (ll)((outl[i]).y + .5) + (ll)((outs[i]).x +
.5);
        res[i] = ((av % MOD * cut + bv) % MOD * cut + cv) %
MOD;
    }
    return res;
}

```

6.10 min25 sieve

```

ll min25_sieve(ll n) {
    // given n, calculate prefix sums of some
multiplicative function f

```

```

// at all points of type floor(n/k) in  $O(n^{3/4}/\log(n))$ , n up to  $1e11$  is ok
// in particular you can find  $f(1) + \dots + f(n)$ 
// also, calculation can be done for primes only, i.e
prefix sum of  $f(i)*I\{i \text{ is prime}\}$ 
// to do that, do not run last stage of algorithm

```

```

vector<ll> v;
v.reserve((int) sqrt(n) * 2 + 7);
ll sq = 0;
{
    ll k = 1;
    while (k * k <= n) {
        v.push_back(k);
        ++k;
    }
    --k;
    sq = k;
    if (k * k == n) --k;
    while (k >= 1) {
        v.push_back(n / k);
        --k;
    }
}
auto geti = [&](ll x) {
    // returns i, such that  $v[i] = x$ 
    if (x <= sq) return x - 1;
    return (int) v.size() - (n / x);
};
// OP1:  $f(ab) = f(a)f(b)$  for coprime a, b;  $f(p) = p^T$ ;
 $f(p^k)$  can be calculated in  $O(1)$ ; we denote  $f(p^k) = g(p, k)$  (p is prime) for all k
// OP2: f also can be any fully multiplicative
function,  $f(ab) = f(a)f(b)$  for all a,b; you need to
calc pref sum of f fast, so only prime case is
useful

auto g = [&](ll p, int k) {
    if (k == 1) {
        return p - 1; // polynomial, for primes-only can
be any fully multiplicative function
    }
    return p + k; // any function,  $g(p^k)$ 
};

auto f = [&](ll x) {
    return g(x, 1);
};

auto pref = [&](ll x) {
    // return  $\sum_{i=1..x} g(i, 1)$ , i.e  $1^T + 2^T + \dots + x^T$ 
    return x * (x + 1) / 2;
};

```

```

vector<ll> s0(v.size()), s1(v.size()); // for all
degrees separately
for (int i = 0; i < (int) v.size(); i++) {
    s0[i] = v[i] % M;

```

```

s1[i] = (((v[i] % M) * ((v[i] + 1) % M) % M) * ((M
+ 1) / 2) % M)) % M; // pref for  $g(p, 1)$ , degrees
separately
//  $s[i] = \text{pref}(v[i]) - 1$  for primes
}

vector<ll> used_primes;
used_primes.reserve((int) sqrt(n) + 7);
for (ll p = 2; p * p <= n; ++p) {
    if (s0[p - 1] == s0[p - 2]) continue;
    // p is prime
    used_primes.push_back(p);
    for (int i = (int) v.size() - 1; i >= 0; --i) {
        if (v[i] < p * p) break; // very important, dont
remove!
        s0[i] += M - ((s0[geti(v[i] / p)] + M - s0[p - 2])
% M * (1)) % M; //  $p^0$ 
        s0[i] %= M;
        s1[i] += M - ((s1[geti(v[i] / p)] + M - s1[p - 2])
% M * (p)) % M; //  $p^1$ 
        s1[i] %= M;
        //  $s[i] += M - ((s[geti(v[i] / p)] + M - s[p-2]) \% M * f(p)) \% M;$ 
    }
}

// PRIMES ONLY calculation is done
// desired answer for  $v[i]$  is in  $s[i]$ 
// in particular  $\sum_{i=1}^n f(i)*I\{i \text{ is prime}\}$  is in
s.back()
// now last stage for default calculation

vector<ll> s(v.size());
for (int i = 0; i < v.size(); i++) {
    s[i] = (M - s0[i] % M + s1[i]) % M; // combine
polynomial by degrees with needed coeffs
}

vector<ll> r = s;

for (int ui = (int) used_primes.size() - 1; ui >= 0;
--ui) { // ui >= 1, sum for odd numbers only
    ll p = used_primes[ui];
    for (int i = (int) v.size() - 1; i >= 0; --i) {
        if (v[i] < p * p) break; // very important, dont
remove!
        for (ll c = 1, pc = p; pc * p <= v[i]; c++, pc *=
p) { //  $pc = p^c$ 
            r[i] += g(p, c + 1) % M + ((g(p, c) % M) * ((M +
r[geti(v[i] / pc)] - s[geti(p)]) % M)) % M;
            r[i] %= M;
        }
    }
}

// done, answer for  $v[i]$  is  $r[i]+1$  ( $f(1)=1$ )
// in particular  $\sum_{i=2}^n f(i)$  is in  $r.back()$ 
// therefore  $\sum_{i=1}^n f(i)$  is  $r.back() + 1$ 
return r.back() + 1 - g(1, 1); // since  $f(1)=1$  for
real, not  $g(1,1)$ : 1 is not prime

```

}

6.11 sqrt mod

```
// p is prime
// -1 if no solution
//  $x = \sqrt{a, p} \implies x^2 = a$  and  $(-x)^2 = a$ 
//  $O(\log n)$  if  $p \equiv 3 \pmod 4$  else  $O(\log^2 n)$ 
// should be changed if const p
ll sqrt(ll a, ll p) {
    a %= p;
    if (a < 0) a += p;
    if (a == 0) return 0;
    if (binpow(a, (p - 1) / 2, p) != 1)
        return -1; // no solution
    if (p % 4 == 3) return binpow(a, (p + 1) / 4, p);
    ll s = p - 1, n = 2;
    int r = 0, m;
    while (s % 2 == 0) ++r, s /= 2;
    while (binpow(n, (p - 1) / 2, p) != p - 1) ++n;
    ll x = binpow(a, (s + 1) / 2, p);
    ll b = binpow(a, s, p), g = binpow(n, s, p);
    for (; r = m) {
        ll t = b;
        for (m = 0; m < r && t != 1; ++m) t = t * t % p;
        if (m == 0) return x;
        ll gs = binpow(g, 1LL << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p;
    }
}
```

6.12 Faycc

```
vector<vector<int>> gauss(vector<vector<int>> &a) {
    int n = a.size();
    int m = a[0].size();
    // int det = 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]);
                if (i != row) {
                    // det *= -1;
                }
                break;
            }
        }
        if (!a[row][col])
            continue;
        for (int i = 0; i < n; ++i) {
            if (i != row && a[i][col]) {
                int val = a[i][col] * inv(a[row][col]) % mod;
                for (int j = col; j < m; ++j) {
                    a[i][j] -= val * a[row][j];
                }
            }
        }
    }
}
```

```
        a[i][j] %= mod;
    }
}
++row;
}
// for (int i = 0; i < n; ++i) det = (det * a[i][i]) %
// mod;
// det = (det % mod + mod) % mod;
// result in (-mod, mod)
return a;
}

pair<int, vector<int>> sle(vector<vector<int>> a, vector
<int> b) {
    int n = a.size();
    int m = a[0].size();
    assert(n == b.size());
    for (int i = 0; i < n; ++i) {
        a[i].push_back(b[i]);
    }
    a = gauss(a);
    vector<int> x(m, 0);
    for (int i = n - 1; i >= 0; --i) {
        int leftmost = m;
        for (int j = 0; j < m; ++j) {
            if (a[i][j] != 0) {
                leftmost = j;
                break;
            }
        }
        if (leftmost == m && a[i].back() != 0) return {-1,
        {}};
        if (leftmost == m) continue;
        int val = a[i].back();
        for (int j = m - 1; j > leftmost; --j) {
            val -= a[i][j] * x[j];
            val %= mod;
        }
        x[leftmost] = (val * inv(a[i][leftmost]) % mod + mod
        ) % mod;
    }
    return {1, x};
}
```

```
vector<bitset<N>> gauss_bit(vector<bitset<N>> a, int m)
{
    int n = a.size();
    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;
        for (int i = 0; i < n; ++i)
            if (i != row && a[i][col])
```

```
        a[i] ^= a[row];
        ++row;
    }
    return a;
}
```

6.13 Диофантовы уравнения

```
//  $ax + by = \pm gcd$  if  $a < 0$  or  $b < 0$ 
pair<int, int> ext_gcd(int a, int b) {
    int x1 = 1, y1 = 0, x2 = 0, y2 = 1;
    while (b) {
        int k = a / b;
        x1 -= x2 * k;
        y1 -= y2 * k;
        a %= b;
        swap(x1, x2), swap(y1, y2), swap(a, b);
    }
    return {x1, y1};
}

// solve  $ax + by = c$  with minimum  $x \geq 0$ 
bool cool_ext_gcd(int a, int b, int c, int &x, int &y) {
    if (b == 0) {
        y = 0;
        if (a == 0)
            return x = 0, c == 0;
        return x = c / a, c % a == 0;
    }
    auto [x0, y0] = ext_gcd(a, b);
    int g = (ll)x0 * a + (ll)y0 * b;
    if (c % g != 0) return false;
    x = (ll)x0 * (c / g) % (b / g);
    if (x < 0) x += abs(b / g);
    y = (c - (ll)a * x) / b;
    return true;
}
```

6.14 KTO

```
// ans % p_i = a_i
vector<vector<int>> r(k, vector<int>(k));
for (int i = 0; i < k; ++i)
    for (int j = 0; j < k; ++j)
        if (i != j)
            r[i][j] = binpow(p[i] % p[j], p[j] - 2, p[j]); //
            [phi(p[j]) - 1] для не простого модуля
vector<int> x(k);
for (int i = 0; i < k; ++i) {
    x[i] = a[i];
    for (int j = 0; j < i; ++j) {
        x[i] = r[j][i] * (x[i] - x[j]);
        x[i] = x[i] % p[i];
        if (x[i] < 0) x[i] += p[i];
    }
}
int ans = 0;
```

```
for (int i = 0; i < k; ++i) {
    int val = x[i];
    for (int j = 0; j < i; ++j) val *= p[j];
    ans += val;
}
```

6.15 Код Грея

```
for (int i = 0; i < (1 << n); i++) {
    gray[i] = i ^ (i >> 1);
}
```

6.16 Линейное решето

```
const int N = 10000000;
int lp[N + 1];
vector<int> pr;
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];
}
```

6.17 Миллер Рабин

```
// works for all n < 2^64
const ll MAGIC[7] = {2, 325, 9375, 28178, 450775,
    9780504, 1795265022};

bool is_prime(ll n) {
    if (n == 1) return false;
    if (n <= 3) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    ll s = __builtin_ctzll(n - 1), d = n >> s; //
        n - 1 = 2^s · d
    for (auto a : MAGIC) {
        if (a % n == 0) {
            continue;
        }
        ll x = binpow(a, d, n); // a -> __int128 in binpow
        for (int _ = 0; _ < s; _++) {
            ll y = binpow(x, 2, n); // x -> __int128 in
            binpow
            if (y == 1 && x != 1 && x != n - 1) {
                return false;
            }
            x = y;
        }
        if (x != 1) {
            return false;
        }
    }
}
```

```
return true;
}
```

6.18 Мёбиус

```
vector<int> mu(n + 1);
mu[1] = 1;
for (int x = 1; x <= n; x++) {
    for (int y = x + x; y <= n; y += x) mu[y] -= mu[x];
}
```

6.19 Подсчёт прогулок

```
int count_walks_1(int b1, int b2, int p, int q) {
    // counting walks from (0, 0) to (p, q)
    // each turn x += 1 or y += 1
    // without touching y = x + b1 and y = x + b2
    // b1 < 0 < b2 must hold
    // 0 < (p + q) / (b2 - b1)
    if (min(p, q) < 0) return 0;
    ll ans = C(p, p + q);
    ar(2) F = {p, q}, S = {p, q};
    int cf = mod - 1;
    while (true) {
        F[1] -= b1;
        swap(F[0], F[1]);
        F[1] += b1;
        S[1] -= b2;
        swap(S[0], S[1]);
        S[1] += b2;
        swap(F, S);
        int wf = C(F[0], F[0] + F[1]);
        int ws = C(S[0], S[0] + S[1]);
        ans += (cf * (ll) ((wf + ws) % mod)) % mod;
        if (wf == 0 && ws == 0) break;
        cf = mod - cf;
    }
    ans %= mod;
    return (int) ans;
}
```

```
int count_walks_2(int b1, int b2, int p, int q) {
    // counting walks from (0, 0) to (p, q)
    // each turn x += 1 and (y -= 1 or y += 1)
    // without touching y = b1 and y = b2
    // b1 < 0 < b2 must hold
    // 0 < (p / (b2 - b1))
    if (abs(p) % 2 != abs(q) % 2) return 0;
    int p0 = (p - q) / 2, q0 = (p + q) / 2;
    return count_walks_1(b1, b2, p0, q0);
}
```

6.20 Ро-Поллард

```
typedef long long ll;
```

```
ll mult(ll a, ll b, ll mod) {
    return (__int128)a * b % mod;
}

ll f(ll x, ll c, ll mod) {
    return (mult(x, x, mod) + c) % mod;
}

ll rho(ll n, ll x0=2, ll c=1) {
    ll x = x0;
    ll y = x0;
    ll g = 1;
    while (g == 1) {
        x = f(x, c, n);
        y = f(y, c, n);
        y = f(y, c, n);
        g = gcd(abs(x - y), n);
    }
    return g;
}
```

```
mt19937_64 rnd(time(nullptr));
```

```
void factor(int n, vector<int> &pr) {
    if (n == 4) {
        factor(2, pr);
        factor(2, pr);
        return;
    }
    if (n == 1) {
        return;
    }
    if (is_prime(n)) {
        pr.push_back(n);
        return;
    }
    int d = rho(n, rnd() % (n - 2) + 2, rnd() % 3 + 1);
    factor(n / d, pr);
    factor(d, pr);
}
```

6.21 Чудо Формулы

- $\sum_{0 \leq k \leq n} \binom{n-k}{k} = \text{Fib}_{n+1}$
- $\sum_{i=0}^k (-1)^i \binom{n}{i} = (-1)^k \binom{n-1}{k}$
- $\sum_{i=0}^k \binom{n+i}{i} = \binom{n+k+1}{k}$
- $\sum_{k=0}^r \binom{m}{k} \binom{n}{r-k} = \binom{m+n}{r}$
- $\sum_{i=r}^n \binom{i}{r} = \binom{n+1}{r+1}$
- $2B(l, r, n) = \binom{n}{l} + \binom{n}{r} - \binom{n+1}{l} + B(l, r, n+1)$, где $B(l, r, n) = \sum_{i=l}^r \binom{n}{i}$
- Степень p в $\binom{n}{m}$ равна числу переносов при сложении m и $n-m$ в p -ичной системе.
- $\sum_{i=0}^n \frac{\binom{k}{i}}{\binom{n}{i}} = \frac{\binom{n+1}{n-k+1}}{\binom{n}{k}}$

9. Число перестановок без фиксированных точек:
 $d(n) = (n-1)(d(n-1) + d(n-2))$, $d(0) = 1$, $d(1) = 0$
10. $\binom{m}{n} \equiv \prod_i \binom{m_i}{n_i} \pmod{p}$, где m_i, n_i — цифры m, n в p -ичной записи.
11. $\sum_{i=0}^n \binom{n}{i} i^k = \sum_{j=0}^k S_2(k, j) \frac{n!}{(n-j)!} 2^{n-j}$
12. $\sum_{i=0}^{n-1} \binom{i}{j} x^i = x^j (1-x)^{-j-1} \left(1 - x^n \sum_{i=0}^j \binom{n}{i} x^{j-i} (1-x)^i \right)$
13. $P(n) = \sum_{k=0}^n \binom{n}{k} Q(k) \implies Q(n) = \sum_{k=0}^n (-1)^{n-k} \binom{n}{k} P(k)$
14. $P(n) = \sum_{k=0}^n (-1)^k \binom{n}{k} Q(k) \implies Q(n) = \sum_{k=0}^n (-1)^k \binom{n}{k} P(k)$
15. $N(n, k)$ — число ПСП длины $2n$ с $2k$ блоками $N(n, k) = \binom{n}{k-1} * \binom{n}{k} / n$
16. $S_1(n, k)$ — число перестановок длины n с k циклами, $S_1(n, k) = (n-1)S_1(n-1, k) + S_1(n-1, k-1)$, $S_1(0, 0) = 1$
17. $S_2(n, k)$ — число разбиений n -множества на k непустых блоков, $S_2(n, k) = kS_2(n-1, k) + S_2(n-1, k-1)$, $S_2(0, 0) = 1$
18. $S_2(n, 2) = 2^{n-1} - 1$
 $S_2(n, 3) = \frac{1}{2}(3^{n-1} - 2^{n-1}) - \frac{1}{2}(3^{n-1} - 1)$
 $S_2(n, 4) = \frac{1}{2!} \left[(4^{n-1} - 3^{n-1}) - (4^{n-1} - 2^{n-1}) + \frac{1}{3}(4^{n-1} - 1) \right]$
 $S_2(n, 5) = \frac{1}{24} 5^{n-1} - \frac{1}{6} 4^{n-1} + \frac{1}{4} 3^{n-1} - \frac{1}{6} 2^{n-1} + \frac{1}{24}$
19. $S_2^d(n, k)$ — число разбиений с расстоянием $\geq d$ внутри блока, $S_2^d(n, k) = S_2(n-d+1, k-d+1)$
20. $\sum_{i=1}^n i a^i = \frac{a(na^{n+1} - (n+1)a^n + 1)}{(a-1)^2}$
21. $\prod_{n=1}^{\infty} (1-x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k (x^{k(3k-1)/2} + x^{k(3k+1)/2})$
22. n — Фибоначчи \iff хотя бы одно из чисел $5n^2 \pm 4$ — точный квадрат.
23. Период последовательности Фибоначчи по модулю n не превышает $6n$.
24. Пифагоровы тройки: пусть $m > n > 0$, $\gcd(m, n) = 1$, и не оба нечётные. Тогда $a = k(m^2 - n^2)$, $b = k(2mn)$, $c = k(m^2 + n^2)$
25. $\#\{(a, b) : a^2 + b^2 = n^2, a > 0, b > 0\} = \left(\prod_{p^\alpha \parallel n, p \equiv 4k+1} 2\alpha + 1 \right) - 1$
26. $r_4(n) = \#\{x_1^2 + \dots + x_4^2 = n\}$, $r_8(n) = \#\{x_1^2 + \dots + x_8^2 = n\}$
 $r_4(n) = 8 \sum_{d|n, 4 \nmid d} d$, $r_8(n) = 16 \sum_{d|n} (-1)^{n+d} d^3$
27. $\#\{ax + by = n, x, y \geq 0\} = \frac{n}{ab} - \left\{ \frac{b'n}{a} \right\} - \left\{ \frac{a'n}{b} \right\} + 1$,
где a' и b' — обратные: $aa' \equiv 1 \pmod{b}$, $bb' \equiv 1 \pmod{a}$
28. $\varphi(mn) = \varphi(m)\varphi(n)\frac{d}{\varphi(d)}$, $d = \gcd(m, n)$
29. $n^x \bmod m = n^{\varphi(m) + (x \bmod \varphi(m))} \bmod m$, $x \geq \log_2 m$
30. $\sum_{n=1}^N \mu^2(n) = \sum_{k=1}^{\lfloor \sqrt{N} \rfloor} \mu(k) \lfloor N/k^2 \rfloor$
31. $g(n) = \sum_{d|n} f(d) \implies f(n) = \sum_{d|n} \mu(d) g(n/d)$
32. $F(n) = \prod_{d|n} f(d) \implies f(n) = \prod_{d|n} F(n/d)^{\mu(d)}$
33. $\gcd(a, \text{lcm}(b, c)) = \text{lcm}(\gcd(a, b), \gcd(a, c))$

34. $\text{lcm}(a, \gcd(b, c)) = \gcd(\text{lcm}(a, b), \text{lcm}(a, c))$
35. $\gcd(\text{lcm}(a, b), \text{lcm}(b, c), \text{lcm}(a, c)) = \text{lcm}(\gcd(a, b), \gcd(b, c), \gcd(a, c))$
36. $\sum_{i,j=1}^n \text{lcm}(i, j) = \sum_{l=1}^n \left(\frac{(1+\lfloor n/l \rfloor) \lfloor n/l \rfloor}{2} \right)^2 \sum_{d|l} \mu(d) l d$
37. $\sum_{i=1}^n \text{lcm}(i, n) = \frac{n}{2} \left(\sum_{d|n} \varphi(d) d + 1 \right)$
38. $\left(\frac{p}{q} \right) \left(\frac{q}{p} \right) = (-1)^{\frac{(p-1)(q-1)}{4}}$ для нечётных простых p, q
39. $\varphi(m+1)$ — кол-во бинарных строк с m различными подпоследовательностями.

7 Строки

7.1 Z-функция

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

7.2 eertree

```
int len[MAXN], suf[MAXN];
int go[MAXN][ALPH];
char s[MAXN];

int n, last, sz;

void init() {
    n = 0, last = 0;
    s[n++] = -1;
    suf[0] = 1; // root of suflink tree = 1
    len[1] = -1;
    sz = 2;
}

int get_link(int v) {
    while (s[n - len[v] - 2] != s[n - 1])
```

```
        v = suf[v];
    return v;
}

void add_char(char c) {
    c -= 'a';
    s[n++] = c;
    last = get_link(last);
    if (!go[last][c]) {
        len[sz] = len[last] + 2;
        suf[sz] = go[get_link(suf[last])][c];
        go[last][c] = sz++;
    }
    last = go[last][c]; // cur v = last
}
```

7.3 Ахо-Корасик

```
int go[MAXN][ALPH];
vector<int> term[MAXN];
int par[MAXN], suf[MAXN];
char par_c[MAXN];
vector<int> g[MAXN];

int cntv = 1;

void add(string &s) {
    static int cnt_s = 1;
    int v = 0;
    for (char el: s) {
        if (go[v][el - 'a'] == 0) {
            go[v][el - 'a'] = cntv;
            par[cntv] = v;
            par_c[cntv] = el;
            cntv++;
        }
        v = go[v][el - 'a'];
    }
    term[v].push_back(cnt_s++);
}

void bfs() {
    deque<int> q = {0};
    while (!q.empty()) {
        int v = q.front();
        q.pop_front();
        if (v > 0) {
            if (par[v] == 0) {
                suf[v] = 0;
            } else {
                suf[v] = go[suf[par[v]]][par_c[v] - 'a'];
            }
            g[suf[v]].push_back(v);
        }
        for (int c = 0; c < ALPH; c++) {
            if (go[v][c] == 0) {
                go[v][c] = go[suf[v]][c];
            } else {
                q.push_back(go[v][c]);
            }
        }
    }
}
```

```

    }
  }
}

```

7.4 Муффиксный Сассив

```

vector<int> build_suff_arr(string &s) {
    // Remove, if you want to sort cyclic shifts
    s += (char) (1);
    int n = s.size();
    vector<int> a(n);
    iota(all(a), 0);
    stable_sort(all(a), [&](int i, int j) {
        return s[i] < s[j];
    });
    vector<int> c(n);
    int cc = 0;
    for (int i = 0; i < n; i++) {
        if (i == 0 || s[a[i]] != s[a[i - 1]])
            c[a[i]] = cc++;
        else
            c[a[i]] = c[a[i - 1]];
    }
    for (int L = 1; L < n; L *= 2) {
        vector<int> cnt(n);
        for (auto i: c) cnt[i]++;
        if (*min_element(all(cnt)) > 0) break;
        vector<int> pref(n);
        for (int i = 1; i < n; i++)
            pref[i] = pref[i - 1] + cnt[i - 1];
        vector<int> na(n);
        for (int i = 0; i < n; i++) {
            int pos = (a[i] - L + n) % n;
            na[pref[c[pos]]++] = pos;
        }
        a = na;
        vector<int> nc(n);
        cc = 0;
        for (int i = 0; i < n; i++) {
            if (i == 0 || c[a[i]] != c[a[i - 1]] ||
                c[(a[i] + L) % n] != c[(a[i - 1] + L) % n])
                nc[a[i]] = cc++;
            else
                nc[a[i]] = nc[a[i - 1]];
        }
        c = nc;
    }
    // Remove, if you want to sort cyclic shifts
    a.erase(a.begin());
    s.pop_back();
    return a;
}

vector<int> kasai(string s, vector<int> sa) {
    // lcp[i] = lcp(sa[i], sa[i + 1])
    int n = s.size(), k = 0;
    vector<int> lcp(n, 0);
    vector<int> rank(n, 0);

```

```

    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    for (int i = 0; i < n; i++, k ? k-- : 0) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = sa[rank[i] + 1];
        while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
        lcp[rank[i]] = k;
    }
    return lcp;
}

```

7.5 Префикс-функция

```

vector<int> prefix_func(string s) {
    int n = s.size();
    vector<int> pref(n, 0);
    int ans = 0;
    for (int i = 1; i < n; i++) {
        while (ans > 0 && s[ans] != s[i]) {
            ans = pref[ans - 1];
        }
        if (s[i] == s[ans]) {
            ans++;
        }
        pref[i] = ans;
    }
    return pref;
}

```

7.6 Суффиксный автомат

// Суфавтомат с подсчётом кол-ва различных подстрок

```

const int SIGMA = 26;
int ans = 0;

struct Node {
    int go[SIGMA];
    int s, p;
    int len;

    Node() {
        fill(go, go + SIGMA, -1);
        s = -1, p = -1;
        len = 0;
    }
};

int add(int A, int ch, vector<Node> &sa) {
    int B = sa.size();
    sa.emplace_back();
    sa[B].p = A;
    sa[B].s = 0;
    sa[B].len = sa[A].len + 1;

```

```

for (; A != -1; A = sa[A].s) {
    if (sa[A].go[ch] == -1) {
        sa[A].go[ch] = B;
        continue;
    }
    int C = sa[A].go[ch];
    if (sa[C].p == A) {
        sa[B].s = C;
        break;
    }
    int D = sa.size();
    sa.emplace_back();
    sa[D].s = sa[C].s;
    sa[D].p = A;
    sa[D].len = sa[A].len + 1;
    sa[C].s = D;
    sa[B].s = D;
    copy(sa[C].go, sa[C].go + SIGMA, sa[D].go);
    for (; A != -1 && sa[A].go[ch] == C; A = sa[A].s)
        sa[A].go[ch] = D;
    break;
}
ans += sa[B].len - sa[sa[B].s].len;
return B;
}

signed main() {
    string s;
    cin >> s;
    vector<Node> sa(1);
    int A = 0;
    for (char c : s)
        A = add(A, c - 'a', sa);
    cout << ans;
}

```

8 Структуры данных

8.1 Disjoint Sparse Table

```

// MAXN дополнить до степени двойки (или n*2)
int tree[LOG][MAXN];
int floorlog2[MAXN]; // i ? (31 - __builtin_clz(i)) : 0

void build(vector<int> &a) {
    int n = a.size();
    copy(a.begin(), a.end(), tree[0]);
    for (int lg = 1; lg < LOG; ++lg) {
        int len = 1 << lg;
        auto &lvl = tree[lg];
        for (int m = len; m < n; m += len * 2) {
            lvl[m - 1] = a[m - 1];
            lvl[m] = a[m];
            for (int i = m - 2; i >= m - len; --i)
                lvl[i] = min(lvl[i + 1], a[i]);
            for (int i = m + 1; i < m + len && i < n; ++i)
                lvl[i] = min(lvl[i - 1], a[i]);
        }
    }
}

```



```

}
for (int i = 2; i < min(MAXN, n * 2); ++i)
    floorlog2[i] = floorlog2[i / 2] + 1;
}

// a[l..r)
int get(int l, int r) {
    r--;
    int i = floorlog2[l ^ r];
    return min(tree[i][l], tree[i][r]);
}

```

8.2 Segment Tree Beats

```

// %=, =, sum
// mx[v], all_equal[v]
// break: mx[v] < x
// tag: all_equal[v] == true, запрос становится =mx[v]%x

// min=, max=, =, +=, sum, mn, mx
// также как и для min=, sum
// для max= храним mn[v], sec_mn[v]

// +=, gcd
// храним gcd разностей какого-то остовного дерева
// храним any_value[v] = любое значение на отрезке
// gcd(l..r) = gcd(any_value[v], gcd[v])
// при слиянии добавляем к gcd значение |a_v[l] - a_v[r]|

// min=, sum
struct ST {
    vector<int> st, mx, mx_cnt, sec_mx;

    ST(vector<int> &a) {
        int n = a.size();
        st.resize(n * 4), mx.resize(n * 4);
        mx_cnt.resize(n * 4, 0), sec_mx.resize(n * 4, 0);
        build(0, 0, n, a);
    }

    void upd_from_children(int v) {
        st[v] = st[v * 2 + 1] + st[v * 2 + 2];
        mx[v] = max(mx[v * 2 + 1], mx[v * 2 + 2]);
        mx_cnt[v] = 0;
        sec_mx[v] = max(sec_mx[v * 2 + 1], sec_mx[v * 2 + 2]);
        if (mx[v * 2 + 1] == mx[v]) {
            mx_cnt[v] += mx_cnt[v * 2 + 1];
        } else {
            sec_mx[v] = max(sec_mx[v], mx[v * 2 + 1]);
        }
        if (mx[v * 2 + 2] == mx[v]) {
            mx_cnt[v] += mx_cnt[v * 2 + 2];
        } else {
            sec_mx[v] = max(sec_mx[v], mx[v * 2 + 2]);
        }
    }
}

```

```

void build(int i, int l, int r, vector<int> &a) {
    if (l + 1 == r) {
        st[i] = mx[i] = a[l];
        mx_cnt[i] = 1;
        sec_mx[i] = -INF;
        return;
    }
    int m = (r + 1) / 2;
    build(i * 2 + 1, l, m, a);
    build(i * 2 + 2, m, r, a);
    upd_from_children(i);
}

void push_min_eq(int v, int val) {
    if (mx[v] > val) {
        st[v] -= (mx[v] - val) * mx_cnt[v];
        mx[v] = val;
    }
}

void push(int i, int l, int r) {
    if (l + 1 < r) {
        push_min_eq(i * 2 + 1, mx[i]);
        push_min_eq(i * 2 + 2, mx[i]);
    }
}

void update(int i, int l, int r, int ql, int qr, int val) {
    if (qr <= l || r <= ql || mx[i] <= val) {
        return;
    }
    if (ql <= l && r <= qr && sec_mx[i] < val) {
        push_min_eq(i, val);
        return;
    }
    push(i, l, r);
    int m = (l + r) / 2;
    update(i * 2 + 1, l, m, ql, qr, val);
    update(i * 2 + 2, m, r, ql, qr, val);
    upd_from_children(i);
}

int sum(int i, int l, int r, int ql, int qr) {
    if (qr <= l || r <= ql) {
        return 0;
    }
    push(i, l, r);
    if (ql <= l && r <= qr) {
        return st[i];
    }
    int m = (l + r) / 2;
    return sum(i * 2 + 1, l, m, ql, qr) + sum(i * 2 + 2, m, r, ql, qr);
}
};

```

8.3 ДД по неявному

```

// потому что nds[0].sz == 0 и sz не изменяется в push
int size(int t) { return nds[t].sz; }

pair<int, int> split(int t, int k) {
    if (!t) return {0, 0};
    push(t);
    int szl = size(nds[t].l);
    if (k <= szl) {
        auto [l, r] = split(nds[t].l, k);
        nds[t].l = r;
        pull(t);
        return {l, t};
    } else {
        auto [l, r] = split(nds[t].r, k - szl - 1);
        nds[t].r = l;
        pull(t);
        return {t, r};
    }
}

// всё остальное ровно как в обычном ДД
// не забыть обновлять sz в pull
// инициализация sz=0 в Node() и sz=1 в Node(...)

```

8.4 ДД

```

// insert (key, val), erase key, max(val) for key ∈ [l, r),
// val+= for key ∈ [l, r)
struct Node {
    int l, r;
    int x, y;
    int val, mx, mod;

    // value of empty set
    Node() : val(-INF), mx(-INF) {
        l = r = 0, mod = 0;
    }
    Node(int x, int val) : x(x), val(val), mx(val) {
        l = r = 0, mod = 0, y = rnd();
    }
};

Node nds[MAX];
int ndsz = 1; // nds[0] means empty

void push(int t) {
    if (!t || nds[t].mod == 0) return;
    nds[t].val += nds[t].mod;
    nds[t].mx += nds[t].mod;
    if (nds[t].l) nds[nds[t].l].mod += nds[t].mod;
    if (nds[t].r) nds[nds[t].r].mod += nds[t].mod;
    nds[t].mod = 0;
}

int getmx(int t) {
    push(t); // delete if sure (faster)
    return nds[t].mx;
}

```

```

void pull(int t) {
    if (!t) return;
    push(t), push(nds[t].l), push(nds[t].r); // must have
    nds[t].mx = max(nds[t].val, max(getmx(nds[t].l), getmx(
        nds[t].r)));
}

pair<int, int> split(int t, int x) {
    if (!t) return {0, 0};
    push(t);
    if (x <= nds[t].x) {
        auto [l, r] = split(nds[t].l, x);
        nds[t].l = r;
        pull(t);
        return {l, t};
    } else {
        auto [l, r] = split(nds[t].r, x);
        nds[t].r = l;
        pull(t);
        return {t, r};
    }
}

int merge(int l, int r) {
    push(l), push(r);
    if (!l) return r;
    if (!r) return l;
    if (nds[l].y < nds[r].y) {
        nds[l].r = merge(nds[l].r, r);
        pull(l);
        return l;
    } else {
        nds[r].l = merge(l, nds[r].l);
        pull(r);
        return r;
    }
}

void insert(int &root, int x, int val) {
    nds[ndsz++] = Node(x, val);
    auto [l, r] = split(root, x);
    root = merge(merge(l, ndsz - 1), r);
}

// erase all equal to x
void erase(int &root, int x) {
    auto [lm, r] = split(root, x + 1);
    auto [l, m] = split(lm, x);
    root = merge(l, r);
}

// query [l, r]
int query(int &root, int ql, int qr) {
    auto [lm, r] = split(root, qr);
    auto [l, m] = split(lm, ql);
    int res = getmx(m);
    root = merge(merge(l, m), r);
    return res;
}

```

```

// update [l, r]
void update(int &root, int ql, int qr, int qx) {
    auto [lm, r] = split(root, qr);
    auto [l, m] = split(lm, ql);
    if (m) nds[m].mod += qx;
    root = merge(merge(l, m), r);
}

```

8.5 Персистентное ДД по неявному

```

struct Node;
int size(int);
int sum(int);

struct Node {
    int l, r;
    int val, sz, sm;

    Node() : val(0), sz(0), sm(0) {}
    Node(int val, int l, int r) : val(val), l(l), r(r) {
        sz = 1 + size(l) + size(r);
        sm = val + sum(l) + sum(r);
    }
};

Node nds[MAX];
int ndsz = 1;

int size(int t) { return nds[t].sz; }
int sum(int t) { return nds[t].sm; }

int newNode(int val, int l, int r) {
    nds[ndsz++] = Node(val, l, r);
    return ndsz - 1;
}

pair<int, int> split(int t, int x) {
    if (!t) return {0, 0};
    int szl = size(nds[t].l);
    if (szl >= x) {
        auto [l, r] = split(nds[t].l, x);
        int v = newNode(nds[t].val, r, nds[t].r);
        return {l, v};
    } else {
        auto [l, r] = split(nds[t].r, x - szl - 1);
        int v = newNode(nds[t].val, nds[t].l, l);
        return {v, r};
    }
}

bool chooseleft(int szl, int szr) {
    return rnd() % (szl + szr) < szl;
}

int merge(int l, int r) {
    if (!l) return r;
    if (!r) return l;
    if (chooseleft(nds[l].sz, nds[r].sz)) {

```

```

        int rr = merge(nds[l].r, r);
        int v = newNode(nds[l].val, nds[l].l, rr);
        return v;
    } else {
        int ll = merge(l, nds[r].l);
        int v = newNode(nds[r].val, ll, nds[r].r);
        return v;
    }
}

```

```

int insert(int root, int ponds[t].s, int val) {
    int new_v = newNode(val, 0, 0);
    auto [l, r] = split(root, pos);
    return merge(merge(l, new_v), r);
}

```

```

int erase(int root, int pos) {
    auto [lm, r] = split(root, pos + 1);
    auto [l, m] = split(lm, pos);
    return merge(l, r);
}

```

```

// query [l, r]
pair<int, int> query(int root, int ql, int qr) {
    auto [lm, r] = split(root, qr);
    auto [l, m] = split(lm, ql);
    int res = sum(m);
    auto new_root = merge(merge(l, m), r);
    return {res, new_root};
}

```

8.6 Персистентное ДО

```

Node nds[MAX];
int ndsz = 1;
// nds[0] is default (empty) value

int sum(int v) { return nds[v].sm; }

// returns new root of subtree
int update(int v, int l, int r, int qi, int qx) {
    if (qi < l || r <= qi) return v;
    if (l + 1 == r) {
        nds[ndsz++] = Node(qx);
        return ndsz - 1;
    }
    int m = (l + r) / 2;
    int u = ndsz++;
    nds[u].l = update(nds[v].l, l, m, qi, qx);
    nds[u].r = update(nds[v].r, m, r, qi, qx);
    nds[u].sm = sum(nds[u].l) + sum(nds[u].r);
    return u;
}

int get(int v, int l, int r, int ql, int qr) {
    if (!v || qr <= l || r <= ql) return 0;
    if (ql <= l && r <= qr) return nds[v].sm;
    int m = (l + r) / 2;
    auto a = get(nds[v].l, l, m, ql, qr);

```

```

    auto b = get(nds[v].r, m, r, ql, qr);
    return a + b;
}

```

8.7 Спарсы

```

int tree[LOG][MAXN];
int floorlog2[MAXN]; // i ? (31 - __builtin_clz(i)) : 0

void build(vector<int> &a) {
    int n = a.size();
    copy(a.begin(), a.end(), tree[0]);
    for (int i = 1; i < LOG; ++i) {
        int len = 1 << (i - 1);
        for (int j = 0; j + len < n; ++j)
            tree[i][j] = min(tree[i - 1][j], tree[i - 1][j + len]);
    }
    for (int i = 2; i <= n; ++i)
        floorlog2[i] = floorlog2[i / 2] + 1;
}

// min a[l..r)
int get(int l, int r) {
    int i = floorlog2[r - l];
    return min(tree[i][l], tree[i][r - (1 << i)]);
}

```

8.8 Фенвик (+ на отрезке)

```

// a[l..r) += x
void update(int l, int r, int x) {
    T1.add(l, x);
    T1.add(r, -x);
    T2.add(l, -x * l);
    T2.add(r, x * r);
}

// sum a[0..i)
int get(int i) {
    return T1.get(i) * i + T2.get(i);
}

```

8.9 Фенвик

```

// Нумерация с 0

struct Fenwick {
    int n;
    vector<int> f;

    Fenwick(int n) : n(n) {
        f.resize(n + 1);
    }

    // a[i] += x

```

```

void add(int i, int x) {
    for (++i; i <= n; i += i & -i)
        f[i] += x;
}

// sum a[0..i)
int get(int i) {
    int ans = 0;
    for (; i > 0; i -= i & -i)
        ans += f[i];
    return ans;
}

// a[l..r) > 0; find max k: sum a[0..k) <= x
int max_not_more(int x) {
    int cur = 0;
    for (int i = 20; i >= 0; --i) {
        int len = 1 << i;
        if (cur + len <= n && f[cur + len] <= x) {
            cur += len;
            x -= f[cur];
        }
    }
    return cur;
};

// sum a[x1..x2)[y1..y2)[z1..x2)
int sum_3d(int x1, int x2, int y1, int y2, int z1, int z2) {
    int ans = get(x2, y2, z2);
    ans -= get(x1, y2, z2) + get(x2, y1, z2) + get(x2, y2, z1);
    ans += get(x1, y1, z2) + get(x1, y2, z1) + get(x2, y1, z1);
    ans -= get(x1, y1, z1);
    return ans;
}

```