

## Muffix Sassif – TRD

Andrianov, Lepeshov, Shulyatev

November 16, 2024



## Содержание

<b>1 Геометрия</b>	<b>2</b>		<b>8</b>		<b>18</b>
1.1 3D	2	2.7 KCC	8	6.9 Гаусс	18
1.2 База 1: вектор	2	2.8 Минкост (Джонсон)	9	6.10 Диофантовы уравнения	19
1.3 База 2: прямая	3	2.9 Мосты	9	6.11 КТО	19
1.4 База 3: окружность	3	2.10 Паросочетания	9	6.12 Код Грея	19
1.5 Выпуклая оболочка	3	2.11 Точки сочленения	9	6.13 Линейное решето	19
1.6 Задача 16	3	2.12 Эдмондс-Карп	10	6.14 Миллер Рабин	19
1.7 Касательные из точки	4	2.13 Эйлеров цикл	10	6.15 Мёбиус	19
1.8 Касательные параллельные прямой	4			6.16 Ро-Поллард	19
1.9 Лежит ли точка в многоугольнике	4	<b>3 ДП</b>	<b>10</b>	<b>7 Строки</b>	<b>20</b>
1.10 Минимальная покрывающая окружность	4	3.1 СHT	10	7.1 Z-функция	20
1.11 Пересечение полуплоскостей	5	3.2 Li Chao	11	7.2 eertree	20
1.12 Проверка на пересечение отрезков	5	3.3 SOS-dp	11	7.3 Ахо-Корасик	20
1.13 Сумма Минковского	5	3.4 НВП	11	7.4 Муффиксный Сассив	20
1.14 Формула Эйлера	5	3.5 НОВП	11	7.5 Префикс-функция	21
<b>2 Графы</b>	<b>5</b>	<b>4 Деревья</b>	<b>11</b>	7.6 Суффиксный автомат	21
2.1 2-SAT	5	4.1 Centroid	11	<b>8 Структуры данных</b>	<b>21</b>
2.2 WeightedMatching	5	4.2 HLD	12	8.1 Disjoint Sparse Table	21
2.3 l-r-capacity-maxflow	7	4.3 Link-cut	12	8.2 Segment Tree Beats	21
2.4 Венгерский алгоритм	7	<b>5 Другое</b>	<b>13</b>	8.3 ДД по неявному	22
2.5 Вершинная двусвязность	7	5.1 Fast mod	13	8.4 ДД	22
2.6 Диниц	8	5.2 Slope trick	13	8.5 Персистентное ДД по неявному	23
		5.3 attribute_packed	13	8.6 Персистентное ДО	23
		5.4 custom_bitset	13	8.7 Спарсы	23
		5.5 ordered_set	14	8.8 Фенвик (+ на отрезке)	23
		5.6 pragma	14	8.9 Фенвик	24
		5.7 Аллокатор Копелиовича	14		
		5.8 Альфа-бета отсечение	14		
		5.9 Отжиг	14		
		<b>6 Математика</b>	<b>14</b>		
		6.1 A:B cmp C:D	14		
		6.2 FFT mod	14		
		6.3 FFT	16		
		6.4 Floor Sum	16		
		6.5 berlekamp-massey-and-friends	16		
		6.6 convMod	17		
		6.7 min25 sieve	17		
		6.8 sqrt mod	18		

# 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 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)};
}

```

### 1.8 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;
}

```

### 1.9 Лежит ли точка в многоугольнике

```

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

```

### 1.10 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;
}

```

### 1.11 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);
    }
    return pts;
}

```

### 1.12 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;
}

```

### 1.13 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];
    return C;
}

```

### 1.14 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

```

for (int i = 1; i <= n; ++i) {
    not_v[i] = i + n;
    not_v[i + n] = i;
}
for (int i = 0; i < m; ++i) {
    cin >> u >> v;
    g[not_v[v]].push_back(u);
    g[not_v[u]].push_back(v);
    rg[u].push_back(not_v[v]);
    rg[v].push_back(not_v[u]);
}
// делаем КСС, получаем comp
for (int v = 1; v <= n; ++v) {
    if (comp[v] == comp[not_v[v]]) {
        cout << "UNSATISFIABLE\n";
        return 0;
    }
}
for (int v = 1; v <= n; ++v)
    cout << (comp[v] > comp[not_v[v]] ? v : not_v[v]);

```

### 2.2 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};
}
}

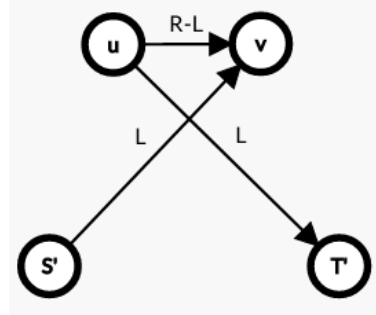
```

### 2.3 l-r-capacity-maxflow

## Maximum flow problem with minimum capacities

We describe how to find the maximum flow from  $S$  to  $T$  when the edges also constrain the minimum bound of the flow amount (edges have “minimum capacities”). It can be boiled down to an ordinary max-flow problem.

Consider an edge from  $u$  to  $v$  whose capacity is  $R$  and minimum capacity is  $L$ . To deal with the minimum capacity, create a new vertex  $S'$  to  $T'$ , remove the original edge, and add edges with the following capacities:



Flow network example with minimum capacities

Add such edges for all edges with the minimum capacities. On the resulting graph, accumulate maximum flow in the following order:

- from  $S'$  to  $T'$
- from  $S'$  to  $T$
- from  $S$  to  $T'$
- from  $S$  to  $T$

An  $S$ – $T$  flow that satisfies the minimum capacities exists if and only if, for all outgoing edges from  $S'$  and incoming edges to  $T'$ , the flow and capacity are equal. (This can be understood by corresponding the flows from  $S'$  and  $T'$  to the original edges.)

Alternatively, if you just want to know the existence of a flow satisfying the minimum capacities, one can add an edge from  $T$  to  $S$  with infinite capacity and consider the flow from  $S'$  to  $T'$  once, instead of accumulating flows four times.

## 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) x (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

```

struct edge {
    int u, ind;

    bool operator< (const edge &other) const {
        return u < other.u;
    }
};

```

```
vector<int> stack_;
```

```

void paint(int v, int pr = -1) {
    used[v] = pr;
    up[v] = tin[v] = ++timer;
    for (auto e: g[v]) {
        if (e.u == pr) {
            continue;
        }
    }
}

```

```

if (!used[e.u]) {
    stack_.push_back(e.ind);
    paint(e.u, v);
    if (up[e.u] >= tin[v]) {
        ++mx_col;
        while (true) {
            int cur_edge = stack_.back();
            col[cur_edge] = mx_col;
            stack_.pop_back();
            if (cur_edge == e.ind) {
                break;
            }
        }
        up[v] = min(up[v], up[e.u]);
    } else if (tin[e.u] < tin[v]) {
        stack_.push_back(e.ind);
        up[v] = min(up[v], tin[e.u]);
    } else if (up[v] > tin[e.u]) {
        up[v] = up[e.u];
    }
}

signed main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        g[u].push_back({v, i});
        g[v].push_back({u, i});
    }
    for (int v = 1; v <= n; ++v) {
        sort(all(g[v]));
    }
    for (int v = 1; v <= n; ++v) {
        if (!used[v]) {
            paint(v);
        }
    }
    for (int v = 1; v <= n; ++v) {
        int len = g[v].size();
        for (int i = 1; i < len; ++i) {
            if (col[g[v][i].ind] == 0) {
                col[g[v][i].ind] = col[g[v][i - 1].ind];
            }
        }
    }
}

```

## 2.6 Диниц

```

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

```

```

struct Dinic {
    vector<Edge> graph[MAXN];
    // разные битсеты, чтобы легко найти разрез

```

```

bitset<MAXN> vis, visf;
int dist[MAXN], Q[MAXN];
int ql, qr, S, T, BIT;
Dinic() {}

bool bfs() {
    vis.reset();
    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;
            // удаление ифа может ускорить,
            // но скорее сильно замедлит
            if (u == T) return true;
        }
    }
    return vis[T];
}

int dfs(int v, int minC) {
    if (v == T) return minC;
    visf[v] = true;
    int ans = 0;
    for (auto &e : graph[v]) {
        auto cc = min(minC - ans, e.c - e.f);
        // удалить второе условие, если один битсет
        if (dist[e.u] <= dist[v] || !vis[e.u] || visf[e.u]
            || cc < BIT)
            continue;
        auto f = dfs(e.u, cc);
        if (f != 0) {
            e.f += f, ans += f;
            graph[e.u][e.r].f -= f;
        }
    }
    return ans;
}

void run(int s, int t) {
    S = s, T = t;
    for (BIT = (1ll << LOG); BIT > 0; BIT >>= 1) {
        while (bfs()) {
            visf.reset();
            dfs(S, MAXC);
        }
    }
}

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

```

```

graph[u].push_back({v, 0, c, (int)graph[v].size() - 1});
}
};

void use_example() {
    Dinic dinic;
    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);

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

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

signed main() {
    vector<int> topsort;
    for (int v = 1; v <= n; ++v)
        if (!used[v])
            dfs1(v, topsort);
    reverse(all(topsort));
    for (int j = 1; j <= n; ++j)
        if (!comp[topsort[j - 1]])
            dfs2(topsort[j - 1], j);
}

```



## 2.8 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;
```

```
}
```

## 2.9 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, v);
            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;
        }
    }
}
```

## 2.10 Паросочетания

```
int 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);
    for (int i = 0; i < n; ++i) {
        ans += dfs(i, i + 1);
    }
}
```

## 2.11 Точки сочленения

```
void dfs(int v, int par) {
    vis[v] = 1;
    up[v] = tin[v] = timer++;
    int child = 0;
    for (auto u : g[v]) {
```

```

    if (!vis[u]) {
        dfs(u, v);
        up[v] = min(up[v], up[u]);
        if (up[u] >= tin[v] && par != -1) {
            points.insert(v);
        }
        child++;
    } else if (u != par) {
        up[v] = min(up[v], tin[u]);
    }
}
if (par == -1 && child >= 2) {
    points.insert(v);
}
}
}

```

## 2.12 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;
    }
}

```

## 2.13 Эйлеров цикл

```

// Эйлеров путь/цикл в компоненте связности s. Возвращает
// т.индексы рёбер. Если пути/цикла нет, то алгоритм найдёт
// хуйню.
// 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);
        }
    }
    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 СХТ

```

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 = {OLL, -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 HBII

```

// 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 HOBI

```

// 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][i]);
    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 1.5-1.9 раз быстрее)
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m((-1ULL / b) {}
    ull reduce(ull a) const {
        ull r = a - (ull)((__uint128_t(m) * a) >> 64) * b;
        return r; // r in [0, 2b) // x1.9 speed
        return r >= b ? r - b : r; // x1.5 speed
    }
}; // Usage:
// FastMod F(m);
// ull x_mod_m = F.reduce(x);

```

### 5.2 Slope trick

```

// Дан массив  $a_n$ . Сделать минимальное кол-во  $\pm 1$ , 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 A:B cmp C:D

```

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 FFT mod

```

const int MOD = 998244353; // 7 · 17 · 223 + 1
const int G = 3;
//const int MOD = 7340033; // 7 · 220 + 1
//const int G = 5;
//const int MOD = 469762049; // 7 · 226 + 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 a, int n) {
    int ans = 1;
    while (n) {
        if (n & 1)
            ans = mul(ans, a);
        a = mul(a, a);
        n >>= 1;
    }
}

```

```

    return ans;
}

int inv(int a) {
    return binpow(a, MOD - 2);
}

void initNTT() {
    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) {
    if (!nttinit) {
        initNTT();
        nttinit = 1;
    }
    int lg = log2(n);
    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) {
        for (int i = 0; i < n; i += 2 * len) {
            for (int j = 0; j < len; ++j) {
                int u = a[i + j], v = mul(W[j << num], a[i + j + len]);
                a[i + j] = add(u, v);
                a[i + j + len] = sub(u, v);
            }
        }
        --num;
    }
    if (rev) {
        int rev_n = binpow(n, MOD - 2);
        for (int i = 0; i < n; ++i) a[i] = mul(a[i], rev_n);
        reverse(a.begin() + 1, a.end());
    }
}

vector<int> conv(vector<int> a, vector<int> b) {
    int lg = 0;
    while ((1 << lg) < a.size() + b.size() + 1)
        ++lg;
    int n = 1 << lg;
    assert(a.size() + b.size() <= n + 1);
    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; --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; --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;
}

```

### 6.3 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
}

```

### 6.4 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;
}

```

### 6.5 berlekamp-massey-and-friends

```

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 = (vector<ll>) convMod<MOD>(p, f);
        auto q2 = (vector<ll>) convMod<MOD>(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);
}

```

```

// vals - initials values of recurrence, c - result of
// berlekamp on vals
int getk(vector<ll> &vals, vector<ll> c, ll k) {
    int d = (int) c.size() - 1;
    c[0] = MOD - 1;
    while (c.back() == 0) {

```

```

    c.pop_back();
}
for (auto &el: c) {
    el = (MOD - el % MOD) % MOD;
}
vector<ll> p(d);
copy(vals.begin(), vals.begin() + d, p.begin());
p = (vector<ll>) convMod<MOD>(p, c);
p.resize(d);
return getkfps(p, c, k);
}

vector<ll> mod_poly_slow(vector<ll> &p, vector<ll> &q) {
    int n = (int) p.size(), m = (int) q.size();
    if (n < m) {
        return p;
    }
    vector<ll> r = p;
    for (int i = n - 1; i >= m - 1; --i) {
        int coef = (r[i] * 1ll * inverse(q[m - 1])) % MOD;
        for (int j = 0; j < m; ++j) {
            r[i - j] = (r[i - j] - coef * 1ll * q[m - 1 - j] % MOD + MOD) % MOD;
        }
    }
    while (!r.empty() && r.back() == 0) {
        r.pop_back();
    }
    return r;
}

int kth_term(vector<ll> &a, vector<ll> s, ll k) {
    // a in 0-index, s in 1-index
    int n = (int) s.size() - 1;
    assert((int) a.size() >= n);
    // a[0]..a[n-1] -- first values
    // a[i] = a[i-1] \cdot s[1] + \ldots + a[i-n] \cdot s[n]
    // find a[k]

    // return getk(a, s, k);

    vector<ll> f(n + 1);
    for (int i = 1; i <= n; ++i) {
        f[i] = (MOD - s[i]) % MOD;
    }
    f[n] = 1;
    vector<ll> res = {1}, w = {0, 1};
    while (k) {
        if (k % 2) {
            res = (vector<ll>) convMod<MOD>(res, w); // mul
        }
        res = mod_poly_slow(res, f);
        w = (vector<ll>) convMod<MOD>(w, w); // mul
        w = mod_poly_slow(w, f);
        k /= 2;
    }
    int ans = 0;

```

```

    for (int i = 0; i < (int) res.size(); i++) {
        ans = (ans + a[i] * 1ll * res[i] % MOD) % MOD;
    }
    return ans;
}

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, cur;
    int lf = 0, d = 0;
    for (int i = 0; i < a.size(); ++i) {
        ll t = 0;
        for (int j = 0; j < cur.size(); ++j) {
            t = (t + 1ll * a[i - j - 1] * cur[j]) % MOD;
        }
        if ((t - a[i]) % MOD == 0) continue;
        if (cur.empty()) {
            cur.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() < cur.size())
            c.resize(cur.size());
        for (int j = 0; j < cur.size(); ++j) {
            c[j] = (c[j] + cur[j]) % MOD;
        }
        if (i - lf + (int) ls.size() >= (int) cur.size()) {
            tie(ls, lf, d) = make_tuple(cur, i, (t - a[i]) % MOD);
        }
        cur = c;
    }
    cur.insert(cur.begin(), 0); // fictive s[0] = 0
    for (auto &i: cur)
        i = (i % MOD + MOD) % MOD;
    return cur;
}

```

## 6.6 convMod

```

typedef complex<double> comp;

template<int M>
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(M));
    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] + conj(L[j])) * R[i] / (2.0 * n)
;
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n)
/ 1i;
    }
    fft(outl, lg), fft(outs, lg);
    for (int i = 0; i < res.size(); i++) {
        ll av = ll(real(outl[i]) + .5), cv = ll(imag(
outs[i]) + .5);
        ll bv = ll(imag(outl[i]) + .5) + ll(real(outs[i]
)) + .5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) %
M;
    }
    return res;
}

```

## 6.7 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 0(n^{3/4}/log
    (n)), n up to 1e11 is ok
    // in particular you can find f(1) + ... + f(n)
    // also, calculation can be done for primes only, i.
    // e prefix sum of f(i)*I{i 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 + ... + 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] = 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 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
```

## 6.8 sqrt mod

```
// -1 if no solution
// x=sqrt(a, p) => x^2=a and (-x)^2=a;
// 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;
    }
```

```
}
}
```

## 6.9 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;
}

```

## 6.10 Диофантовы уравнения

```

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 = x1 - x2 * k;
        y1 = y1 - y2 * k;
        swap(x1, x2);
        swap(y1, y2);
        a %= b;
        swap(a, b);
    }
    return {x1, y1};
}

```

```

bool cool_ext_gcd(int a, int b, int c, int &x, int &y) {
    if (b == 0) {
        y = 0;
        if (a == 0) {
            x = 0;
            return c == 0;
        } else {
            x = c / a;

```

```

        return c % a == 0;
    }
}

auto [x0, y0] = ext_gcd(a, b);
int g = x0 * a + y0 * b;
if (c % g != 0)
    return false;
x0 *= c / g;
y0 *= c / g;
int t = b / g;
int k = (-x0) / t;
if (x0 + t * k < 0)
    k += t / abs(t);
x = x0 + t * k;
y = y0 - (a / g) * k;
return true;
}

```

## 6.11 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]);
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;
}

```

## 6.12 Код Грея

```

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

```

## 6.13 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];
}

```

## 6.14 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;
}

```

## 6.15 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];
}

```

## 6.16 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);
}

```

## 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 < MAXN; ++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 сливании 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

```

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

```

```

Node *merge(Node *l, Node *r) {
    if (!l)
        return r;
    if (!r)
        return l;
    if (l->y < r->y) {
        l->r = merge(l->r, r);
        pull(l);
        return l;
    } else {
        r->l = merge(l, r->l);
        pull(r);
        return r;
    }
}

```

```

void insert(Node *&root, int pos, int val) {
    Node *new_v = new Node(val);
    auto [l, r] = split(root, pos);
    root = merge(merge(l, new_v), r);
}

```

```

}

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

int sum(Node *v) {
    return v ? v->sm : 0;
}

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

```

### 8.4 ДД

```

pair<Node *, Node *> split(Node *t, int x) {
    if (!t)
        return {nullptr, nullptr};
    if (x <= t->x) {
        auto [l, r] = split(t->l, x);
        t->l = r;
        pull(t);
        return {l, t};
    } else {
        auto [l, r] = split(t->r, x);
        t->r = l;
        pull(t);
        return {t, r};
    }
}

```

```

Node *merge(Node *l, Node *r) {
    if (!l)
        return r;
    if (!r)
        return l;
    if (l->y < r->y) {
        l->r = merge(l->r, r);
        pull(l);
        return l;
    } else {
        r->l = merge(l, r->l);
        pull(r);
        return r;
    }
}

```

```

void insert(Node *&root, int val) {
    Node *new_v = new Node(val);
    auto [l, r] = split(root, val);
}

```

```

    root = merge(merge(l, new_v), r);
}

void erase(Node *&root, int val) {
    auto [lm, r] = split(root, val + 1);
    auto [l, m] = split(lm, val);
    root = merge(l, r);
}

int sum(Node *v) {
    return v ? v->sm : 0;
}

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

```

## 8.5 Персистентное ДД по неявному

mt19937 rnd(228);

```

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

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

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

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

int sum(Node *v) {
    return v ? v->sm : 0;
}

pair<Node *, Node *> split(Node *t, int x) {
    if (!t)
        return {nullptr, nullptr};
    int lsz = size(t->l);
    if (lsz >= x) {
        auto [l, r] = split(t->l, x);
        auto v = new Node(t->val, r, t->r);

```

```

        return {l, v};
    } else {
        auto [l, r] = split(t->r, x - lsz - 1);
        auto v = new Node(t->val, t->l, l);
        return {v, r};
    }
}

bool chooseleft(int lsz, int rsz) {
    return rnd() % (lsz + rsz) < lsz;
}

Node *merge(Node *l, Node *r) {
    if (!l)
        return r;
    if (!r)
        return l;
    if (chooseleft(l->sz, r->sz)) {
        auto rr = merge(l->r, r);
        auto v = new Node(l->val, l->l, rr);
        return v;
    } else {
        auto ll = merge(l, r->l);
        auto v = new Node(r->val, ll, r->r);
        return v;
    }
}

Node *insert(Node *root, int pos, int val) {
    Node *new_v = new Node(val);
    auto [l, r] = split(root, pos);
    return merge(merge(l, new_v), r);
}

Node *erase(Node *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, Node *> query(Node *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

```

// left: v ? v->l : nullptr (same for right)
// sum: v ? v->sm : 0

// v can be nullptr. returns new root of subtree
Node *update(Node *v, int l, int r, int qi, int qx) {
    if (qi < l || r <= qi)
        return v;
    if (l + 1 == r)

```

```

        return new Node(qx);
    int m = (l + r) / 2;
    Node *u = new Node();
    u->l = update(left(v), l, m, qi, qx);
    u->r = update(right(v), m, r, qi, qx);
    u->sm = sum(u->l) + sum(u->r);
    return u;
}

int get(Node *v, int l, int r, int ql, int qr) {
    if (!v || qr <= l || r <= ql)
        return 0;
    if (ql <= l && r <= qr)
        return v->sm;
    int m = (l + r) / 2;
    auto a = get(v->l, l, m, ql, qr);
    auto b = get(v->r, m, r, ql, qr);
    return a + b;
}

```

## 8.7 Спарсы

```

int tree[LOG][MAXN];
int floorlog2[LOG]; // 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..pos]
int rsq(int pos) {
    return T1.rsq(pos) * pos + T2.rsq(pos);
}

```



```

}

// sum a[l..r)
int sum(int l, int r) {
    return rsq(r) - rsq(l);
}

```

## 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[..] > 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;
}

```