Muffix Sassif – TRD

Andrianov, Lepeshov, Shulyatev

Содержание

1 Геометрия

1.1 3D

```
double eps = 1e-7;

struct Pt {
  double x;
  double y;
  double z;

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

  Pt operator-(const Pt& other) const {
    return {x - other.x, y - other.y, z - other.z};
  }

  Pt operator+(const Pt& other) const {
    return {x + other.x, y + other.y, z + other.z};
  }

  Pt operator/(const double& a) const {
    return {x / a, y / a, z / a};
  }

  Pt operator*(const double& a) const {
    return {x * a, y * a, z * a};
  }

  Pt cross(const Pt& p2) const {
    double nx = y * p2.z - z * p2.y;
    double ny = z * p2.x - x * p2.z;
    double nz = x * p2.y - y * p2.x;
    return {nx, ny, nz};
  }

  bool operator==(const Pt& pt) const {
    return abs(x - pt.x) < eps && abs(y - pt.y) < eps &&
      abs(z - pt.z) < eps;
  }

  double dist() {
    return sqrtl(x * x + y * y + z * z);
  }
};

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

  Plane(double a_, double b_, double c_, double d_) : a(
    a_), b(b_), c(c_), d(d_) {
    double kek = sqrtl(a * a + b * b + c * c);
    if (kek < eps) return;
    a /= kek;
    b /= kek;
    c /= kek;
    d /= kek;
  }

  double get_val(Pt p) {
    return a * p.x + b * p.y + c * p.z + d;
  }

  bool on_plane(Pt p) {
    return abs(get_val(p)) / sqrtl(a * a + b * b + c * c
    ) < eps;
  }

  Pt proj(Pt p) {
    double t = (a * p.x + b * p.y + c * p.z + d) / (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<double, double>, pair<double, double>> get_xy(
    double a, double b, double c) {
  if (abs(a) > eps) {
    double y1 = 0, y2 = 10;
    return {{(-c - b * y1) / a, y1}, {(-c - b * y2) / a,
     y2}};
  }
  double 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) {
    double nd = pl1.d - pl1.a * pl2.d / pl2.a;
    double nc = pl1.c - pl1.a * pl2.c / pl2.a;
    double 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);
    double x1 = (-pl2.d - pl2.c * yz1.second - pl2.b *
    yz1.first) / pl2.a;
    double 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)};
}
```

1.2 Вектор, прямая, окружность

```
//// Вектор ////

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;
}

//// 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;
    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 vctr(x, y);
  }
  void normalize() {
    dbl d = sqrtl(a * a + b * b);
    a /= d;
    b /= d;
    c /= d;
  }
};

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

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

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

//// Окружность ////

struct circle {
  dbl x, y, r;

  circle() {}
  circle(dbl x, dbl y, dbl r) : x(x), y(y), r(r) {}
  circle(vctr P, dbl r) : x(P.x), y(P.y), r(r) {}
  circle(const vctr A, const vctr B) {
    vctr C = (A + B) / 2;
    x = C.x, y = C.y;
    r = (A - B).dist() / 2;
  }
  circle(const vctr A, const vctr B, const vctr C) {
    line l1 = bisection(A, B);
    line l2 = bisection(B, C);
    vctr P = intersection(l1, l2);
    x = P.x, y = P.y;
    r = (P - A).dist();
  }
  bool isin(const vctr P) const {
    return (vctr(x, y) - P).dist2() <= r * r;
  }
  vctr cent() const { return vctr(x, y); }
};
```

1.3 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 = rotate_min_vctr(arr);
  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;
}
```

1.4 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));
}
```

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

1.6 Касательные параллельные прямой

```
// find point with max signed distance to line
int tangent_parallel_line(const vector<vctr> &p, line l)
    {
  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;
}
```

1.7 Лежит ли точка в многоугольнике

```
// P starts with 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;
}
```

1.8 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;
}
```

1.9 Многоугольники

```
// Сдвиг многоугольника, чтобы начинался с минимального
    вектора
vctr rotate_min_vctr(vector<vctr> &pts) {
  int ind = 0;
  for (int i = 1; i < pts.size(); ++i) {
    if (is2plane(pts[i] - pts[ind]))
      ind = i;
  }
  rotate(pts.begin(), pts.begin() + ind, pts.end());
  return pts[0];
}

// 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;
}
```

1.10 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;
}
```

1.11 Пересечение с окружностью

```
bool is_intersection_line_circ(line l, circle c) {
  l.normalize();
  dbl d = abs(l.get(c.cent()));
  return d < c.r - EPS;
```

```
}

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

vector<vctr> intersection_circ_circ(circle a, circle b)
    {
  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.cent() - v).dist();
  dbl k = sqrtl(d * d - c.r * c.r);
  circle c2(v.x, v.y, k);
  return intersection_circ_circ(c, c2);
}
```

1.12 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

```
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);
  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 Венгерский алгоритм

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

```
      for (int j = 0; j <= m; ++j)
        if (used[j])
          u[p[j]] += delta, v[j] -= delta;
        else
          minv[j] -= delta;
      j0 = j1;
    } while (p[j0] != 0);
    do {
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    } while (j0);
  }
  int cost = -v[0];
  vector<int> ans(n + 1);
  for (int j = 1; j <= m; ++j)
    ans[p[j]] = j;
  return {cost, ans};
}
```

2.3 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];
      }
    }
  }
}
```

2.4 Дейкстра за квадрат

```
// 0-based
pair<vector<int>, vector<int>> dijkstra(int start, int n
    ) {
  vector<int> dist(n, INF);
  vector<int> pred(n, -1);
  vector<int> used(n);
  dist[start] = 0;
  for (int _ = 0; _ < n; ++_) {
    int v = -1;
    for (int i = 0; i < n; ++i) {
      if (!used[i])
        if (v == -1 || dist[v] > dist[i])
          v = i;
    }
    if (v == -1) break;
    for (auto [u, w] : g[v]) {
      if (dist[u] > w + dist[v]) {
        dist[u] = w + dist[v];
        pred[u] = v;
      }
    }
    used[v] = 1;
  }
  return {dist, pred};
}
```

2.5 Диниц

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

vector<edge> g[MAXN];
pair<int, int> pred[MAXN];
int d[MAXN];
int inds[MAXN];

bool dfs(int v, int final, int W) {
  if (v == final) {
    return true;
  }
  for (int i = inds[v]; i < (int) g[v].size(); i++) {
    auto e = g[v][i];
    if (e.f + W <= e.c && d[v] + 1 == d[e.v]) {
      pred[e.v] = {v, i};
      bool flag = dfs(e.v, final, W);
      if (flag) {
        return true;
      }
      inds[v]++;
    } else {
      inds[v]++;
```

```
    }
  }
  return false;
}

bool bfs(int start, int final, int W) {
  fill(d, d + MAXN, INF);
  d[start] = 0;
  deque<int> q = {start};
  while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto e : g[v]) {
      if (e.f + W <= e.c && d[e.v] > d[v] + 1) {
        d[e.v] = d[v] + 1;
        q.push_back(e.v);
      }
    }
  }
  if (d[final] == INF) {
    return false;
  }
  fill(inds, inds + MAXN, 0);
  while (dfs(start, final, W)) {
    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;
}

void Dinic(int start, int final) {
  int W = (1LL << 30);
  do {
    while (bfs(start, final, W));
    W /= 2;
  } while (W >= 1);
}
```

```
signed main() {
  int n, m;
  vector<pair<int, int>> edges;
  for (int i = 0; i < m; i++) {
    int u, v, c;
    cin >> u >> v >> c;
    edges.emplace_back(u, v);
    g[u].push_back({v, 0, c, (int) g[v].size()});
    // если ребро - ориентированно,
    // то обратная capacity = 0
    g[v].push_back({u, 0, c, (int) g[u].size() - 1});
  }
  int start = 1, target = n;
  Dinic(start, target);
  int res = 0;
  for (auto e : g[start]) {
    res += e.f;
  }
  vector<int> cut;
  for (int i = 0; i < m; i++) {
    int u = edges[i].first, v = edges[i].second;
    if ((d[u] != INF && d[v] == INF) ||
        (d[u] == INF && d[v] != INF)) {
      cut.push_back(i + 1);
    }
  }
}
```

2.6 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);
}
```

2.7 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[MAXM];
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;
}
```

2.8 Мосты

```
void dfs(int v, int par) {
  vis[v] = 1;
  up[v] = tin[v] = timer++;
  for (auto u : g[v]) {
    if (!vis[u]) {
      dfs(u, v);
      up[v] = min(up[v], up[u]);
    } else if (u != par) {
      up[v] = min(up[v], tin[u]);
    }
    if (up[u] > tin[v]) {
      bridges.emplace_back(v, u);
    }
  }
}
```

2.9 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);
  }
}
```

2.10 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);
 }
}
```

2.11 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;
 }
}
```

2.12 Эйлеров цикл

```
// unconnected graph, deleting edges, set<int> g[N];
for (int v = 0; v < n; v++) {
 if (!g[v].empty()) {
  vector<int> ccl;
  vector<int> s = {v};
  while (!s.empty()) {
   int u = s.back();
   if (g[u].empty()) {
    ccl.pb(u);
    s.pop_back();
   } else {
    int u2 = *g[u].begin();
    g[u].erase(u2);
    g[u2].erase(u);
    s.pb(u2);
   }
  }
  // ccl[0] = ccl.back()
  // i.e for graph with edges
  // (1,2),(1,3),(2,3) -> ccl = [1,2,3,1]
 }
}
```

3 ДП

3.1 СНТ

```
struct Line {
 ld k, b;
};

pair<ld, ld> inter(Line a, Line b) {
 ld x = (b.b - a.b) / (a.k - b.k);
 ld y = a.k * x + a.b;
 return {x, y};
}

void add_line(ld k, ld b, vector<Line> &s, vector<pair<
    ld, ld>> &pts) {
 while (s.size() >= 2) {
  pair<ld, ld> x1 = inter(s.back(), s[s.size() - 2]);
  pair<ld, ld> x2 = inter(s[s.size() - 2], {k, b});
  if (x1 > x2) {
   break;
  }
  pts.pop_back();
  s.pop_back();
 }
 if (!s.empty()) {
  pts.push_back(inter(s.back(), {k, b}));
 }
 s.push_back({k, b});
}

ld bin_search(vector<Line> &s, ld x) {
 int l = 0, r = s.size();
 while (l + 1 < r) {
  int m = (r + l) / 2;
  auto kek = inter(s[m - 1], s[m]);
  if (kek.first >= x) {
   l = m;
  } else {
   r = m;
  }
 }
```

```
  return s[l].k * x + s[l].b;
}
```

3.2 Li Chao

```
// MAXIMUM
struct Line {
  int k, b;

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

struct ST {
  vector<Line> st;

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

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

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

3.3 SOS-dp

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

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

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

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

3.4 НВП

```
// 0-indexation ({a_0, ..., a_{n-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 НОВП

```
// 1-indexation ({0, a_1, ..., a_n}, {0, b_1, ..., b_m})
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

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

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

void dfs(int v, int p) {
  // code
  for (auto u : g[v]) {
    if (u != p && !used[u]) {
      dfs(u, v);
    }
  }
}

void solve(int v) {
  sizes(v, -1);
  // code (?)
  for (auto u : g[v]) {
    if (!used[u]) {
      // code (?)
      dfs(u, v);
      // сохраняем результаты dfs
    }
  }
  // code (сливаем результаты dfs)
  used[v] = 1;
  for (int u : g[v]) {
    if (!used[u]) {
      solve(centroid(u, v, sz[u]));
    }
  }
}

signed main() {
  sizes(0, -1);
  solve(centroid(0, -1, n));
}
```

4.2 HLD

```
const int MAXN = 50500;
const int INF = (int) 1e15;
const int L = 20;
vector<int> g[MAXN];
int sz[MAXN];
int depth[MAXN];

vector<vector<int>> up(MAXN, vector<int>(L + 1));

void dfs(int v, int p) {
  up[v][0] = p;
  for (int i = 1; i <= L; i++) {
    up[v][i] = up[up[v][i - 1]][i - 1];
  }
  for (int u : g[v]) {
    if (u != p) {
      dfs(u, v);
    }
  }
}

int lca(int u, int v) {
  if (u == v) {
    return u;
  }
  int du = depth[u], dv = depth[v];
  if (du < dv) {
    swap(du, dv);
    swap(u, v);
  }
  for (int i = L; i >= 0; i--) {
    if (du - (int) pow(2, i) >= dv) {
      u = up[u][i];
      du -= (int) pow(2, i);
    }
  }
  if (u == v) {
    return u;
  }
  for (int i = L; i >= 0; i--) {
    if (up[u][i] != up[v][i]) {
      u = up[u][i];
      v = up[v][i];
    }
  }
  return up[u][0];
}

void dfs1(int v, int p) {
  sz[v] = 1;
  for (int u : g[v]) {
    if (u != p) {
      dfs1(u, v);
      sz[v] += sz[u];
    }
  }
}

int cnt = 0;
int nn[MAXN];
int pred[MAXN];
int rup[MAXN];

void dfs2(int v, int p, int root, int dep = 0) {
  depth[v] = dep;
  nn[v] = cnt++;
  pred[v] = p;
  rup[v] = root;
```

```
  int mx = 0;
  int vert = -1;
  for (int u : g[v]) {
    if (u != p) {
      if (mx < sz[u]) {
        mx = sz[u];
        vert = u;
      }
    }
  }
  if (vert != -1) {
    dfs2(vert, v, root, dep + 1);
  }
  for (int u : g[v]) {
    if (u != p && u != vert) {
      dfs2(u, v, u, dep + 1);
    }
  }
}

ST st({});
int n;

int mx_path_up(int u, int v) {
  if (depth[u] < depth[v]) {
    swap(u, v);
  }
  int res = -INF;
  while (true) {
    int root = rup[u];
    if (depth[root] <= depth[v]) {
      res = max(res, st.rmq(0, 0, n, nn[v], nn[u] + 1));
      break;
    }
    res = max(res, st.rmq(0, 0, n, nn[root], nn[u] + 1))
    ;
    u = pred[rup[u]];
  }
  return res;
}

int mx_path(int u, int v) {
  int vert = lca(u, v);
  return max(mx_path_up(u, vert), mx_path_up(v, vert));
}

void change(int u, int qd) {
  st.update(0, 0, n, nn[u], qd);
}
```

```
signed main() {
  cin >> n;
  vector<int> hs(n);
  for (auto &x : hs) {
    cin >> x;
  }
  for (int i = 0; i < n - 1; i++) {
    cin >> u1 >> v1;
    g[u1].push_back(v1);
    g[v1].push_back(u1);
  }
  dfs1(1, -1);
  dfs(1, 1);
  dfs2(1, -1, 1);
  vector<int> nhs(n);
  for (int i = 1; i <= n; i++) {
    nhs[nn[i]] = hs[i - 1];
  }
  st = *new ST(nhs);
  char op;
  int q;
  cin >> q;
  while (q--) {
    cin >> op >> v1 >> u1;
    if (op == '?') {
      cout << mx_path(u1, v1) << endl;
    } else {
      change(v1, u1);
    }
  }
}
```

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 Slope trick

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

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

5.2 attribute_packed

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

5.3 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);
```

5.4 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")
```

5.5 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) {
  assert((mpos += n) <= 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!!
```

6 Математика

6.1 FFT mod

```
const int MOD = 998244353; // 7 * 17 * 2^23 + 1
const int GEN = 3;
//const int MOD = 7340033; // 7 * 2^20 + 1
//const int GEN = 5;
//const int MOD = 469762049; // 7 * 2^26 + 1
//const int GEN = 30;

const int LOG = 20;
const int MAXN = 1 << LOG;
int tail[MAXN + 1];
int OMEGA[MAXN + 1];

int binpow(int x, int p) {
  int res = 1;
  while (p > 0) {
    if (p & 1)
      res = res * 1ll * x % MOD;
    x = x * 1ll * x % MOD;
    p >>= 1;
  }
  return res;
}

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

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

void calcomega() {
  long long one = binpow(GEN, (MOD - 1) / MAXN);
  OMEGA[0] = 1;
  for (int i = 1; i < MAXN; ++i) {
    OMEGA[i] = OMEGA[i - 1] * one % MOD;
  }
}

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

// Without precalc, tail[], OMEGA[]
//
//long long omega(int n, int k) {
//    return binpow(GEN, (MOD - 1) / n * k);
//}
//
//int gettail(int x, int lg) {
//    int res = 0;
//    for (int i = 0; i < lg; ++i)
//        res += ((x >> i) & 1) << (lg - i - 1);
//    return res;
//}

void fft(vector<int> &A, int lg) {
  int n = 1 << lg;
  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] * 1ll * omega(len, j
    ) % MOD;
        A[i + j] = (v + u) % MOD;
        A[i + j + len / 2] = (v - u + MOD) % MOD;
      }
    }
  }
}

int inverse(int x) {
  return binpow(x, MOD - 2);
}

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

vector<int> mul(vector<int> A, vector<int> B) {
  int lg = 32 - __builtin_clz(A.size() + B.size() - 1);
  int n = 1 << lg;
  A.resize(n, 0);
  B.resize(n, 0);
  fft(A, lg);
  fft(B, lg);
  for (int i = 0; i < n; ++i)
    A[i] = A[i] * 1ll * B[i] % MOD;
  invfft(A, lg);
  return A;
}

signed main() {
  calctail(); // НЕ ЗАБЫТЬ
  calcomega(); // НЕ ЗАБЫТЬ
  int n, m;
  cin >> n >> m;
  vector<int> A(n), B(m);
  for (int &el : A)
    cin >> el;
  for (int &el : B)
    cin >> el;
  auto C = mul(A, B);
  for (auto el : C)
    cout << el << ' ';
}
```

6.2 FFT

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

```
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];
}

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 n = A.size();
  for (int i = 0; i < n; ++i) {
    if (i < tail[i])
      swap(A[i], A[tail[i]]);
  }
  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 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);
  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) {
  fft(A);
  for (auto &el : A)
    el = el / MAXN;
  reverse(A.begin() + 1, A.end());
}

vector<int> mul(vector<int> &a, vector<int> &b) {
  vector<comp> A(MAXN, 0), B(MAXN, 0);
  for (int i = 0; i < (int)a.size(); ++i)
    A[i] = a[i];
  for (int i = 0; i < (int)b.size(); ++i)
    B[i] = b[i];
  fft2(A, B);
  for (int i = 0; i < MAXN; ++i)
    A[i] = A[i] * B[i];
  invfft(A);
  vector<int> c(MAXN);
  for (int i = 0; i < MAXN; ++i) {
    int x = round(A[i].x);
    c[i] = x;
  }
  while (!c.empty() && c.back() == 0)
    c.pop_back();
  return c;
}

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

6.3 Гаусс

```
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;
}
```

6.4 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;
}
```

6.5 КТО

```
// x = a_i % p_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;
}
```

6.6 Код Грея

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

6.7 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];
}
```

6.8 Миллер Рабин

```
// assuming '#define int long long' is ON (replace 'int'
     with 'long long' if not)
// works for all n < 2^64
const int MAGIC[7] = {2, 325, 9375, 28178, 450775,
    9780504, 1795265022};

int bpow(__int128 a, int x, int mod) {
  a %= mod;
  __int128 ans = 1;
  while (x) {
    if (x % 2) {
      ans *= a;
      ans %= mod;
    }
    a *= a;
    a %= mod;
    x /= 2;
  }
  return (int) ans;
}

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

6.9 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 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 < 26; c++) {
      if (go[v][c] == 0) {
        go[v][c] = go[suf[v]][c];
      } else {
        q.push_back(go[v][c]);
      }
    }
  }
}
```

7.3 Муффиксный Сассив

```
vector<int> build_suff_arr(string s) {
  s.push_back('#');
  int n = s.size();
  vector<int> suf(n), c(n);
  vector<int> cnt(MAX);
  for (int i = 0; i < n; i++) {
    cnt[s[i] - '#']++;
  }
  vector<int> pos(MAX);
  for (int i = 1; i < MAX; i++) {
    pos[i] = pos[i - 1] + cnt[i - 1];
  }
  for (int i = 0; i < n; i++) {
    suf[pos[s[i] - '#']++] = i;
  }
  int cls = -1;
  for (int i = 0; i < n; i++) {
    if (i == 0 || s[suf[i]] != s[suf[i - 1]]) {
      cls++;
    }
    c[suf[i]] = cls;
  }
  for (int L = 1; L < n; L *= 2) {
    fill(cnt.begin(), cnt.end(), 0);
    for (int i = 0; i < n; i++) {
      cnt[c[i]]++;
    }
    pos[0] = 0;
    for (int i = 1; i < n; i++) {
      pos[i] = pos[i - 1] + cnt[i - 1];
    }
    for (int i = 0; i < n; i++) {
      suf[i] = (suf[i] - L + n) % n;
    }
    vector<int> new_suf(n), new_c(n);
    for (int i = 0; i < n; i++) {
      int where = pos[c[suf[i]]];
      new_suf[where] = suf[i];
      pos[c[suf[i]]]++;
    }
    cls = -1;
    for (int i = 0; i < n; i++) {
      if (i == 0) {
        cls++;
        new_c[new_suf[i]] = cls;
        continue;
      }
      pair<int, int> prev = {c[new_suf[i - 1]], c[(
    new_suf[i - 1] + L) % n]};
      pair<int, int> now = {c[new_suf[i]], c[(new_suf[i]
     + L) % n]};
      if (prev != now) {
        cls++;
      }
      new_c[new_suf[i]] = cls;
    }
    swap(c, new_c);
    swap(suf, new_suf);
  }
  vector<int> res;
  for (int i = 1; i < n; i++) {
    res.push_back(suf[i]);
  }
  return res;
}

vector<int> kasai(string s, vector<int> sa) {
  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;
}
```

7.4 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;
}
```

7.5 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

```
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 <= n; ++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

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

  ST(int n) {
    st.resize(n * 4, 0);
    mx.resize(n * 4, 0);
    mx_cnt.resize(n * 4, 0);
    sec_mx.resize(n * 4, 0);
    build(0, 0, n);
  }

  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) {
    if (l + 1 == r) {
      st[i] = mx[i] = 0;
      mx_cnt[i] = 1;
      sec_mx[i] = -INF;
      return;
    }
    int m = (r + l) / 2;
    build(i * 2 + 1, l, m);
    build(i * 2 + 2, m, r);
    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) {
    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 (mx[i] <= val) {
      return;
    }
    if (ql == l && qr == r && sec_mx[i] < val) {
      push_min_eq(i, val);
      return;
    }
    push(i);
    int m = (r + l) / 2;
    if (qr <= m) {
      update(i * 2 + 1, l, m, ql, qr, val);
    } else if (ql >= m) {
```

```
      update(i * 2 + 2, m, r, ql, qr, val);
    } else {
      update(i * 2 + 1, l, m, ql, m, val);
      update(i * 2 + 2, m, r, m, qr, val);
    }
    upd_from_children(i);
  }

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

8.3 ДД по неявному

```
pair<Node *, Node *> split(Node *t, int k) {
  if (!now)
    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[MAXN]; // i ? (31 - __builtin_clz(i)) : 0

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

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

8.8 Фенвик (+ на отрезке)

```
// a[l..r) += x
void update(int l, int r, int x) {
```

```
  T1.add(l, x);
  T1.add(r, -x);
  T2.add(l, -x * l);
  T2.add(r, x * r);
}

// sum a[0..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 FenwickTree {
  int n;
  vector<vector<vector<int>>> ft;

  FenwickTree(int n) : n(n) {
    ft.resize(n + 1, vector<vector<int>>(n + 1, vector<
    int>(n + 1)));
  }

  // a[x][y][z] += d
  void upd(int x, int y, int z, int d) {
    x++, y++, z++;
    for (int x1 = x; x1 <= n; x1 += x1 & -x1) {
      for (int y1 = y; y1 <= n; y1 += y1 & -y1) {
        for (int z1 = z; z1 <= n; z1 += z1 & -z1) {
          ft[x1][y1][z1] += d;
        }
      }
    }
  }

  // sum a[0..x)[0..y)[0..z)
  int rsq(int x, int y, int z) {
    int ans = 0;
    for (int x1 = x; x1 > 0; x1 -= x1 & -x1) {
      for (int y1 = y; y1 > 0; y1 -= y1 & -y1) {
        for (int z1 = z; z1 > 0; z1 -= z1 & -z1) {
          ans += ft[x1][y1][z1];
        }
      }
    }
    return ans;
  }

  // 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 = rsq(x2, y2, z2);
    ans -= rsq(x1, y2, z2) + rsq(x2, y1, z2) + rsq(x2,
    y2, z1);
    ans += rsq(x1, y1, z2) + rsq(x1, y2, z1) + rsq(x2,
    y1, z1);
    ans -= rsq(x1, y1, z1);
    return ans;
  }
};
```