1 ComputationalGeometry

1.1 2D

```
Point GetLineIntersection(Point p, Vector v, Point q,
     Vector w) {
  Vector u = p - q;
  double t = Cross(w, u) / Cross(v, w);
  return p + v * t;
}
Point GetLineProjection(Point P, Point A, Point B) {
  Vector v = B - A;
  return A + v * (Dot(v, P - A) / Dot(v, v));
}
double DistanceToLine(Point P, Point A, Point B) {
  Vector v1 = B - A, v2 = P - A;
  return fabs(Cross(v1, v2)) / Length(v1);
}
bool OnSegment(Point p, Point a1, Point a2) {
  return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 -
       p, a2 - p)) < 0;
}
void getLineGeneralEquation(const Point &p1, const Point
     &p2, double &a,
                            double &b, double &c) {
  a = p2.y - p1.y;
  b = p1.x - p2.x;
  c = -a * p1.x - b * p1.y;
}
double DistanceToSegment(Point p, Point a, Point b) {
  if (a == b) return Length(p - a);
  Vector v1 = b - a, v2 = p - a, v3 = p - b;
  if (dcmp(Dot(v1, v2)) < 0)
    return Length(v2);
  else if (dcmp(Dot(v1, v3)) > 0)
    return Length(v3);
  else
    return fabs(Cross(v1, v2)) / Length(v1);
}
double dis_pair_seg(Point p1, Point p2, Point p3, Point p4)
     {
  return min(
         min(DistanceToSegment(p1, p3, p4),
              DistanceToSegment(p2, p3, p4)),
         min(DistanceToSegment(p3, p1, p2),
              DistanceToSegment(p4, p1, p2)));
}
bool SegmentIntersection(Point a1, Point a2, Point b1,
     Point b2) {
  double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1,
       b2 - a1),
       c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1,
            a2 - b1);
  return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}
struct Line {
  Point P;
  Vector v;
  double ang;
  Line() {}
  Line(Point P, Vector v) : P(P), v(v) {
    ang = atan2(v.y, v.x);
  }
  Point point(double a) {
    return p + (v * a);
  }
  bool operator<(const Line &L) const {
    return ang < L.ang;
  }
};
bool OnLeft(const Line &L, const Point &p) {
  return Cross(L.v, p - L.P) > 0;
}
vector<Point> HalfplaneIntersection(vector<Line> L) {
  int n = L.size();
  sort(L.begin(), L.end());
  int first, last;
  vector<Point> p(n);
  vector<Line> q(n);
  vector<Point> ans;
  q[first = last = 0] = L[0];
  for (int i = 1; i < n; i++) {
    while (first < last && !OnLeft(L[i], p[last - 1]))
         last--;
    while (first < last && !OnLeft(L[i], p[first])) first++;
    q[++last] = L[i];
    if (fabs(Cross(q[last].v, q[last - 1].v)) < eps) {
      last--;
      if (OnLeft(q[last], L[i].P)) q[last] = L[i];
    }
    if (first < last)
      p[last - 1] = GetLineIntersection(q[last - 1],
           q[last]);
  }
  while (first < last && !OnLeft(q[first], p[last - 1]))
       last--;
  if (last - first <= 1) return ans;
  p[last] = GetLineIntersection(q[last], q[first]);
  for (int i = first; i <= last; i++) ans.push_back(p[i]);
  return ans;
}
Point PolyGravity(Point *p, int n) {
  Point tmp, g = Point(0, 0);
  double sumArea = 0, area;
  for (int i = 2; i < n; ++i) {
    area = Cross(p[i - 1] - p[0], p[i] - p[0]);
    sumArea += area;
    tmp.x = p[0].x + p[i - 1].x + p[i].x;
    tmp.y = p[0].y + p[i - 1].y + p[i].y;
    g.x += tmp.x * area;
    g.y += tmp.y * area;
  }
  g.x /= (sumArea * 3.0);
  g.y /= (sumArea * 3.0);
  return g;
}
vector<Point> ConvexHull(vector<Point> &p) {
  sort(p.begin(), p.end());
  p.erase(unique(p.begin(), p.end()), p.end());
  int n = p.size();
  int m = 0;
  vector<Point> ch(n + 1);
  for (int i = 0; i < n; i++) {
    while (m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] -
         ch[m - 2]) <= 0)
      m--;
    ch[m++] = p[i];
  }
  int k = m;
  for (int i = n - 2; i >= 0; i--) {
    while (m > k && Cross(ch[m - 1] - ch[m - 2], p[i] -
         ch[m - 2]) <= 0)
      m--;
    ch[m++] = p[i];
  }
  if (n > 1) m--;
  ch.resize(m);
  return ch;
}
int isPointInPolygon(Point p, Polygon poly) {
  int wn = 0;
  int n = poly.size();
  for (int i = 0; i < n; i++) {
    if (OnSegment(p, poly[i], poly[(i + 1) % n])) return -1;
    int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p -
         poly[i]));
    int d1 = dcmp(poly[i].y - p.y);
    int d2 = dcmp(poly[(i + 1) % n].y - p.y);
    if (k > 0 && d1 <= 0 && d2 > 0) wn++;
    if (k < 0 && d2 <= 0 && d1 > 0) wn--;
  }
  if (wn != 0) return 1;
  return 0;
}
int diameter2(vector<Point> &points) {
  vector<Point> p = ConvexHull(points);
  int n = p.size();
  if (n == 1) return 0;
  if (n == 2) return Dist2(p[0], p[1]);
  p.push_back(p[0]);
```

```
  int ans = 0;
  for (int u = 0, v = 1; u < n; u++) {
    for (;;) {
      int diff = Cross(p[u + 1] - p[u], p[v + 1] - p[v]);
      if (diff <= 0) {
        ans = max(ans, Dist2(p[u], p[v]));
        if (diff == 0) ans = max(ans, Dist2(p[u], p[v +
            1]));
        break;
      }
      v = (v + 1) % n;
    }
  }
  return ans;
}
double RC_Distance(Point *ch1, Point *ch2, int n, int m) {
  int q = 0, p = 0;
  REP(i, n) if (ch1[i].y - ch1[p].y < -eps) p = i;
  REP(i, m) if (ch2[i].y - ch2[q].y > eps) q = i;
  ch1[n] = ch1[0];
  ch2[m] = ch2[0];
  double tmp, ans = 1e100;
  REP(i, n) {
    while ((tmp = Cross(ch1[p + 1] - ch1[p], ch2[q + 1] -
         ch1[p]) -
                Cross(ch1[p + 1] - ch1[p], ch2[q] -
                     ch1[p])) > eps)
      q = (q + 1) % m;
    if (tmp < -eps)
      ans = min(ans, DistanceToSegment(ch2[q], ch1[p],
           ch1[p + 1]));
    else
      ans =
        min(ans, dis_pair_seg(ch1[p], ch1[p + 1], ch2[q],
             ch2[q + 1]));
    p = (p + 1) % n;
  }
  return ans;
}
double RC_Triangle(Point *res, int n) {
  if (n < 3) return 0;
  double ans = 0, tmp;
  res[n] = res[0];
  int j, k;
  REP(i, n) {
    j = (i + 1) % n;
    k = (j + 1) % n;
    while ((j != k) && (k != i)) {
      while (Cross(res[j] - res[i], res[k + 1] - res[i]) >
             Cross(res[j] - res[i], res[k] - res[i]))
        k = (k + 1) % n;
      tmp = Cross(res[j] - res[i], res[k] - res[i]);
      if (tmp > ans) ans = tmp;
      j = (j + 1) % n;
    }
  }
  return ans;
}
double fermat_point(Point *pt, int n, Point &ptres) {
  Point u, v;
  double step = 0.0, curlen, explen, minlen;
  int i, j, k, idx;
  bool flag;
  u.x = u.y = v.x = v.y = 0.0;
  REP(i, n) {
    step += fabs(pt[i].x) + fabs(pt[i].y);
    u.x += pt[i].x;
    u.y += pt[i].y;
  }
  u.x /= n;
  u.y /= n;
  flag = 0;
  while (step > eps) {
    for (k = 0; k < 10; step /= 2, ++k)
      for (i = -1; i <= 1; ++i)
        for (j = -1; j <= 1; ++j) {
          v.x = u.x + step * i;
          v.y = u.y + step * j;
          curlen = explen = 0.0;
          REP(idx, n) {
            curlen += dist(u, pt[idx]);
            explen += dist(v, pt[idx]);
          }
          if (curlen > explen) {
            u = v;
            minlen = explen;
            flag = 1;
          }
        }
  }
  ptres = u;
  return flag ? minlen : curlen;
}
bool cmpy(const int &a, const int &b) {
  return point[a].y < point[b].y;
}
double Closest_Pair(int left, int right) {
  double d = INF;
  if (left == right) return d;
  if (left + 1 == right) return dis(left, right);
  int mid = (left + right) >> 1;
  double d1 = Closest_Pair(left, mid);
  double d2 = Closest_Pair(mid + 1, right);
  d = min(d1, d2);
  int i, j, k = 0;
  for (i = left; i <= right; i++) {
    if (fabs(point[mid].x - point[i].x) <= d) tmpt[k++] = i;
  }
  sort(tmpt, tmpt + k, cmpy);
  for (i = 0; i < k; i++) {
    for (j = i + 1; j < k && point[tmpt[j]].y -
         point[tmpt[i]].y < d; j++) {
      double d3 = dis(tmpt[i], tmpt[j]);
      if (d > d3) d = d3;
    }
  }
  return d;
}
int getLineCircleIntersection(Line L, Circle C, double &t1,
    double &t2, vector<Point> &sol) {
  double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y
      - C.c.y;
  double e = a * a + c * c, f = 2 * (a * b + c * d),
       g = b * b + d * d - C.r * C.r;
  double delta = f * f - 4 * e * g;
  if (dcmp(delta) < 0) return 0;
  if (dcmp(delta) == 0) {
    t1 = t2 = -f / (2 * e);
    sol.push_back(L.point(t1));
    return 1;
  }
  t1 = (-f - sqrt(delta)) / (2 * e);
  sol.push_back(L.point(t1));
  t2 = (-f + sqrt(delta)) / (2 * e);
  sol.push_back(L.point(t2));
  return 2;
}
int getCircleCircleIntersection(Circle C1, Circle C2,
    vector<Point> &sol) {
  double d = Length(C1.c - C2.c);
  if (dcmp(d) == 0) {
    if (dcmp(C1.r - C2.r) == 0) return -1;
    return 0;
  }
  if (dcmp(C1.r + C2.r - d) < 0) return 0;
  if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;
  double a = angle(C2.c - C1.c);
  double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2
      * C1.r * d));
  Point p1 = C1.point(a - da), p2 = C1.point(a + da);
  sol.push_back(p1);
  if (p1 == p2) return 1;
  sol.push_back(p2);
  return 2;
}
int getTangents(Point p, Circle C, Vector *v) {
  Vector u = C.c - p;
  double dist = Length(u);
  if (dist < C.r)
    return 0;
  else if (dcmp(dist - C.r) == 0) {
    v[0] = Rotate(u, PI / 2);
```

```
    return 1;
  } else {
    double ang = asin(C.r / dist);
    v[0] = Rotate(u, -ang);
    v[1] = Rotate(u, +ang);
    return 2;
  }
}
int getTangents(Circle A, Circle B, Point *a, Point *b) {
  int cnt = 0;
  if (A.r < B.r) swap(A, B), swap(a, b);
  int d2 =
    (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) *
        (A.c.y - B.c.y);
  int rdiff = A.r - B.r;
  int rsum = A.r + B.r;
  if (d2 < rdiff * rdiff) return 0;
  double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
  if (d2 == 0 && A.r == B.r) return -1;
  if (d2 == rdiff * rdiff) {
    a[cnt] = A.point(base);
    b[cnt] = B.point(base);
    cnt++;
    return 1;
  }
  double ang = acos((A.r - B.r) / sqrt(d2));
  a[cnt] = A.point(base + ang);
  b[cnt] = B.point(base + ang);
  cnt++;
  a[cnt] = A.point(base - ang);
  b[cnt] = B.point(base - ang);
  cnt++;
  if (d2 == rsum * rsum) {
    a[cnt] = A.point(base);
    b[cnt] = B.point(PI + base);
    cnt++;
  } else if (d2 > rsum * rsum) {
    double ang = acos((A.r + B.r) / sqrt(d2));
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(PI + base + ang);
    cnt++;
    a[cnt] = A.point(base - ang);
    b[cnt] = B.point(PI + base - ang);
    cnt++;
  }
  return cnt;
}
Circle CircumscribedCircle(Point p1, Point p2, Point p3) {
  double Bx = p2.x - p1.x, By = p2.y - p1.y;
  double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
  double D = 2 * (Bx * Cy - By * Cx);
  double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx +
      Cy * Cy)) / D + p1.x;
  double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx +
      By * By)) / D + p1.y;
  Point p = Point(cx, cy);
  return Circle(p, Length(p1 - p));
}
Circle InscribedCircle(Point p1, Point p2, Point p3) {
  double a = Length(p2 - p3);
  double b = Length(p3 - p1);
  double c = Length(p1 - p2);
  Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
  return Circle(p, DistanceToLine(p, p1, p2));
}
vector<Point>
    CircleThroughPointTangentToLineGivenRadius(Point p,
    Line L, double r) {
  vector<Point> ans;
  double t1, t2;
  getLineCircleIntersection(L.move(-r), Circle(p, r), t1,
      t2, ans);
  getLineCircleIntersection(L.move(r), Circle(p, r), t1,
      t2, ans);
  return ans;
}
vector<Point> CircleTangentToLinesGivenRadius(Line a, Line
    b, double r) {
  vector<Point> ans;
  Line L1 = a.move(-r), L2 = a.move(r);
  Line L3 = b.move(-r), L4 = b.move(r);
  ans.push_back(GetLineIntersection(L1, L3));
  ans.push_back(GetLineIntersection(L1, L4));
  ans.push_back(GetLineIntersection(L2, L3));
  ans.push_back(GetLineIntersection(L2, L4));
  return ans;
}
vector<Point>
    CircleTangentToTwoDisjointCirclesWithRadius(Circle
    c1, Circle c2, double r) {
  vector<Point> ans;
  Vector v = c2.c - c1.c;
  double dist = Length(v);
  int d = dcmp(dist - c1.r - c2.r - r * 2);
  if (d > 0) return ans;
  getCircleCircleIntersection(Circle(c1.c, c1.r + r),
      Circle(c2.c, c2.r + r), ans);
  return ans;
}
int getSegCircleIntersection(Line L, Circle C, Point *sol) {
  Vector nor = normal(L.v);
  Line pl = Line(C.c, nor);
  Point ip = GetIntersection(pl, L);
  double dis = Length(ip - C.c);
  if (dcmp(dis - C.r) > 0) return 0;
  Point dxy = vecunit(L.v) * sqrt(sqr(C.r) - sqr(dis));
  int ret = 0;
  sol[ret] = ip + dxy;
  if (OnSegment(sol[ret], L.p, L.point(1))) ret++;
  sol[ret] = ip - dxy;
  if (OnSegment(sol[ret], L.p, L.point(1))) ret++;
  return ret;
}
double SegCircleArea(Circle C, Point a, Point b) {
  double a1 = angle(a - C.c);
  double a2 = angle(b - C.c);
  double da = fabs(a1 - a2);
  if (da > PI) da = PI * 2.0 - da;
  return dcmp(Cross(b - C.c, a - C.c)) * da * sqr(C.r) /
      2.0;
}
double PolyCiclrArea(Circle C, Point *p, int n) {
  double ret = 0.0;
  Point sol[2];
  p[n] = p[0];
  REP(i, n) {
    double t1, t2;
    int cnt = getSegCircleIntersection(Line(p[i], p[i + 1]
        - p[i]), C, sol);
    if (cnt == 0) {
      if (!OnOrInCircle(p[i], C) || !OnOrInCircle(p[i + 1],
          C))
        ret += SegCircleArea(C, p[i], p[i + 1]);
      else
        ret += Cross(p[i + 1] - C.c, p[i] - C.c) / 2.0;
    }
    if (cnt == 1) {
      if (OnOrInCircle(p[i], C) && !OnOrInCircle(p[i + 1],
          C))
        ret += Cross(sol[0] - C.c, p[i] - C.c) / 2.0,
              ret += SegCircleArea(C, sol[0], p[i + 1]);
      else
        ret += SegCircleArea(C, p[i], sol[0]),
              ret += Cross(p[i + 1] - C.c, sol[0] - C.c) /
                  2.0;
    }
    if (cnt == 2) {
      if ((p[i] < p[i + 1]) ^ (sol[0] < sol[1]))
        swap(sol[0], sol[1]);
      ret += SegCircleArea(C, p[i], sol[0]);
      ret += Cross(sol[1] - C.c, sol[0] - C.c) / 2.0;
      ret += SegCircleArea(C, sol[1], p[i + 1]);
    }
  }
  return fabs(ret);
}
double area[N];
int n;
struct cp {
  double x, y, r, angle;
  int d;
  cp() {}
```

```
  cp(double xx, double yy, double ang = 0, int t = 0) {
    x = xx;
    y = yy;
    angle = ang;
    d = t;
  }
  void get() {
    scanf("%lf%lf%lf", &x, &y, &r);
    d = 1;
  }
} cir[N], tp[N * 2];
double dis(cp a, cp b) {
  return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}
double cross(cp p0, cp p1, cp p2) {
  return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) *
       (p2.x - p0.x);
}
bool circmp(const cp &u, const cp &v) {
  return dcmp(u.r - v.r) < 0;
}
bool cmp(const cp &u, const cp &v) {
  if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
  return u.d > v.d;
}
double calc(cp cir, cp cp1, cp cp2) {
  double ans = (cp2.angle - cp1.angle) * sqr(cir.r) -
       cross(cir, cp1, cp2) +
            cross(cp(0, 0), cp1, cp2);
  return ans / 2;
}
void CirUnion(cp cir[], int n) {
  cp cp1, cp2;
  sort(cir, cir + n, circmp);
  for (int i = 0; i < n; ++i)
    for (int j = i + 1; j < n; ++j)
      if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r)
           <= 0)
        cir[i].d++;
  for (int i = 0; i < n; ++i) {
    int tn = 0, cnt = 0;
    for (int j = 0; j < n; ++j) {
      if (i == j) continue;
      if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,
           cp2, cp1) < 2)
        continue;
      cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
      cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
      cp1.d = 1;
      tp[tn++] = cp1;
      cp2.d = -1;
      tp[tn++] = cp2;
      if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
    }
    tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
    tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);
    sort(tp, tp + tn, cmp);
    int p, s = cir[i].d + tp[0].d;
    for (int j = 1; j < tn; ++j) {
      p = s;
      s += tp[j].d;
      area[p] += calc(cir[i], tp[j - 1], tp[j]);
    }
  }
}
void solve() {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) cir[i].get();
  memset(area, 0, sizeof(area));
  CirUnion(cir, n);
  for (int i = 1; i <= n; ++i) {
    area[i] -= area[i + 1];
  }
  double tot = 0;
  for (int i = 1; i <= n; i++) tot += area[i];
  printf("%f\n", tot);
}
inline double cross(point o, point a, point b) {
  return (a - o) * (b - o);
}
PDI s[maxN * maxp * 2];
```

```
Polygon P[maxN];
double S, ts;
int N;
inline double seg(point o, point a, point b) {
  if (cmp(b.x - a.x) == 0) return (o.y - a.y) / (b.y - a.y);
  return (o.x - a.x) / (b.x - a.x);
}
double PolygonUnion() {
  int M, c1, c2;
  double s1, s2, ret = 0;
  for (int i = 0; i < N; i++)
    for (int ii = 0; ii < P[i].n; ii++) {
      M = 0;
      s[M++] = mp(0.00, 0);
      s[M++] = mp(1.00, 0);
      for (int j = 0; j < N; j++)
        if (i != j)
          for (int jj = 0; jj < P[j].n; jj++) {
            c1 = cmp(cross(P[i][ii], P[i][ii + 1],
                 P[j][jj]));
            c2 = cmp(cross(P[i][ii], P[i][ii + 1], P[j][jj +
                 1]));
            if (c1 == 0 && c2 == 0) {
              if (((P[i][ii + 1] - P[i][ii]) ^
                   (P[j][jj + 1] - P[j][jj])) > 0 &&
                  i > j) {
                s[M++] = mp(
                         seg(P[j][jj], P[i][ii], P[i][ii +
                              1]), 1);
                s[M++] = mp(
                         seg(P[j][jj + 1], P[i][ii],
                              P[i][ii + 1]),
                         -1);
              }
            } else {
              s1 = cross(P[j][jj], P[j][jj + 1], P[i][ii]);
              s2 = cross(P[j][jj], P[j][jj + 1], P[i][ii +
                   1]);
              if (c1 >= 0 && c2 < 0)
                s[M++] = mp(s1 / (s1 - s2), 1);
              else if (c1 < 0 && c2 >= 0)
                s[M++] = mp(s1 / (s1 - s2), -1);
            }
          }
      sort(s, s + M);
      double pre = min(max(s[0].x, 0.0), 1.0), now;
      double sum = 0;
      int cov = s[0].y;
      for (int j = 1; j < M; j++) {
        now = min(max(s[j].x, 0.0), 1.0);
        if (!cov) sum += now - pre;
        cov += s[j].y;
        pre = now;
      }
      ret += P[i][ii] * P[i][ii + 1] * sum;
    }
  return ret / 2;
}
int main() {
  for (int i = 0; i < N; i++) {
    P[i].n = 4;
    P[i].input();
    ts = P[i].Area();
    if (cmp(ts < 0)) {
      reverse(P[i].p, P[i].p + P[i].n);
      P[i][P[i].n] = P[i][0];
      ts = -ts;
    }
    S += ts;
  }
  printf("%.9lf\n", S / PolygonUnion());
}
// count(c / a + 1, c % a, a, b) + c / a + 1
long long count(long long n, long long a, long long b, long
     long m) {
  if (b == 0) {
    return n * (a / m);
  }
  if (a >= m) {
    return n * (a / m) + count(n, a % m, b, m);
  }
```

```
  if (b >= m) {
    return (n - 1) * n / 2 * (b / m) + count(n, a, b % m,
        m);
  }
  return count((a + b * n) / m, (a + b * n) % m, m, b);
}

bool TriSegIntersection(Point3 P0, Point3 P1, Point3 P2,
    Point3 A, Point3 B, Point3 &P) {
  Vector3 n = Cross(P1 - P0, P2 - P0);
  if (dcmp(Dot(n, B - A)) == 0) return false;
  double t = Dot(n, P0 - A) / Dot(n, B - A);
  if (dcmp(t) < 0 || dcmp(t - 1) > 0) return false;
  P = A + (B - A) * t;
  return PointInTri(P, P0, P1, P2);
}
```

1.2 3D

```
struct Face {
  int v[3];
  Vector3 normal(Point3 *P) const {
    return Cross(P[v[1]] - P[v[0]], P[v[2]] - P[v[0]]);
  }
  int cansee(Point3 *p, int i) const {
    return Dot(P[i] - P[v[0]], normal(P)) > 0 ? 1 : 0;
  }
};
vector<Face> CH3D(Point3 *P, int n) {
  vector<Face> cur;
  cur.push_back((Face) {{0, 1, 2}});
  cur.push_back((Face) {{2, 1, 0}});
  for (int i = 3; i < n; ++i) {
    vector<Face> next;
    for (int j = 0; j < cur.size(); ++j) {
      Face &f = cur[j];
      int res = f.cansee(P, i);
      if (!res) next.push_back(f);
      for (int k = 0; k < 3; ++k) vis[f.v[k]][f.v[(k + 1) %
          3]] = res;
    }
    for (int j = 0; j < cur.size(); ++j)
      for (int k = 0; k < 3; ++k) {
        int a = cur[j].v[k], b = cur[j].v[(k + 1) % 3];
        if (vis[a][b] != vis[b][a] && vis[a][b])
          next.push_back((Face) {{
            a, b, i
          }});
      }
    cur = next;
  }
  return cur;
}
```

2 DataStructure

2.1 BIT

```
int Kth(int K) {
  int ret = 0, i;
  for(i = 21; i >= 0; i--) {
    ret |= (1 << i);
    if(ret >= 200000 || tree[ret] >= K)ret ^= (1 << i);
    else K -= tree[ret];
  }
  return ret + 1;
}
```

2.2 Segment Tree

```
struct Tree {
  int left, right, val, len[11];
};
Tree tree[8 * N];
void PushUp(int ind) {
  int i;
  for(i = 1; i <= K; i++) {
    if(tree[ind].val >= i)
      tree[ind].len[i] = Record[tree[ind].right + 1] -
          Record[tree[ind].left];
    else if(tree[ind].left != tree[ind].right)
      tree[ind].len[i] = tree[L(ind)].len[i -
          tree[ind].val] + tree[R(ind)].len[i -
          tree[ind].val];
    else tree[ind].len[i] = 0;
  }
}
void Update(int ind, int l, int r, int val) {
  if(tree[ind].left == l && tree[ind].right == r) {
    tree[ind].val += val;
    PushUp(ind);
    return ;
  }
  int mid = (tree[ind].left + tree[ind].right) / 2;
  if(r <= mid) Update(L(ind), l, r, val);
  else if(l > mid)Update(R(ind), l, r, val);
  else Update(L(ind), l, mid, val), Update(R(ind), mid + 1,
      r, val);
  PushUp(ind);
}
```

2.3 Scanning Line

```
const int UP = 0; const int DOWN = 1;
const int IN = 0; const int OUT = 1;
vector<int> g[N];
double TT;
struct Circle {
  int x, y, r, w;
  int getX(int flag) {
    return flag == IN? x - r: x + r;
  }
  double getY(int flag) {
    double ret = sqrt(1.0 * r * r - 1.0 * (TT - x) * (TT -
        x));
    return flag == UP? y + ret: y - ret;
  }
} cir[N];
int sgn(double x) {
  if(fabs(x) < 1e-10)return 0;
  return x < 0? -1: 1;
}
struct Line {
  int x, y, idx, flag;
  Line() {}
  Line(int x, int y, int idx, int flag): x(x), y(y),
      idx(idx), flag(flag) {}
  bool operator<(const Line& l)const {
    if(x == l.x) return y > l.y;
    return x < l.x;
  }
} line[N << 1];
struct Node {
  int idx, flag;
  Node() {}
  Node(int idx, int flag):idx(idx), flag(flag) {}
  bool operator<(const Node& n)const {
    double y1 = cir[idx].getY(flag);
    double y2 = cir[n.idx].getY(n.flag);
    int cmp = sgn(y1 - y2);
    if(0 == cmp)return flag < n.flag;
    return cmp > 0;
  }
};
int n, totl;
int pre[N];
void solve() {
  set<Node> re;
  for(int i = 0; i < totl; i++) {
    TT = line[i].x;
    if(OUT == line[i].flag) {
      re.erase(Node(line[i].idx, UP));
```

```
      re.erase(Node(line[i].idx, DOWN));
    } else {
      set<Node>::iterator it = re.insert(Node(line[i].idx,
           UP)).first;
      set<Node>::iterator e = it, f = it;
      e++;
      int idx = it -> idx;
      if(it == re.begin() || e == re.end()) {
        cir[idx].w = 1;
      } else {
        f--;
        if(f -> idx == e->idx) {
          g[f -> idx].push_back(idx);
          cir[idx].w = cir[f -> idx].w + 1;
          pre[idx] = f -> idx;
        } else {
          cir[idx].w = max(cir[f -> idx].w, cir[e -> idx].w);
          if(cir[f -> idx].w >= cir[e -> idx].w && pre[f ->
               idx]) {
            g[pre[f -> idx]].push_back(idx);
            pre[idx] = pre[f -> idx];
          } else if(pre[e -> idx]) {
            g[pre[e -> idx]].push_back(idx);
            pre[idx] = pre[e -> idx];
          }
        }
      }
      re.insert(Node(line[i].idx, DOWN));
    }
  }
}
totl = 0;
for(int i = 0; i < n ; i++) {
  scanf("%d%d%d", &cir[i].x, &cir[i].y, &cir[i].r);
  cir[i].w = 0;
  line[totl++] = Line(cir[i].getX(IN), cir[i].y, i, IN);
  line[totl++] = Line(cir[i].getX(OUT), cir[i].y, i, OUT);
}
sort(line, line + totl);
solve();
```

2.4 2D SegmentTree

```
const int INF = (1 << 30);
struct IntervalTree2D {
  int Max[maxn][maxn], Min[maxn][maxn], n, m;
  int xo, xleaf, x1, y1, x2, y2, x, y, v, vmax, vmin;
  void query1D(int o, int L, int R) {
    if(y1 <= L && R <= y2) {
      vmax = max(Max[xo][o], vmax);
      vmin = min(Min[xo][o], vmin);
    } else {
      int M = L + (R - L) / 2;
      if(y1 <= M)query1D(o * 2, L, M);
      if(M < y2)query1D(o * 2 + 1, M + 1, R);
    }
  }
  void query2D(int o, int L, int R) {
    if(x1 <= L && R <= x2) {
      xo = o;
      query1D(1, 1, m);
    } else {
      int M = L + (R - L) / 2;
      if(x1 <= M)query2D(o * 2, L, M);
      if(M < x2)query2D(o * 2 + 1, M + 1, R);
    }
  }
  void modify1D(int o, int L, int R) {
    if(L == R) {
      if(xleaf) {
        Max[xo][o] = Min[xo][o] = v;
        return ;
      }
      Max[xo][o] = max(Max[xo * 2][o], Max[xo * 2 + 1][o]);
      Min[xo][o] = min(Min[xo * 2][o], Min[xo * 2 + 1][o]);
    } else {
      int M = L + (R - L) / 2;
      if(y <= M)modify1D(o * 2, L, M);
      else modify1D(o * 2 + 1, M + 1, R);
      Max[xo][o] = max(Max[xo][o * 2], Max[xo][o * 2 + 1]);
      Min[xo][o] = min(Min[xo][o * 2], Min[xo][o * 2 + 1]);
    }
  }
  void modify2D(int o, int L, int R) {
    if(L == R) {
      xo = o;
      xleaf = 1;
      modify1D(1, 1, m);
    } else {
      int M = L + (R - L) / 2;
      if(x <= M) {
        modify2D(o * 2, L, M);
      } else {
        modify2D(o * 2 + 1, M + 1, R);
      }
      xo = o;
      xleaf = 0;
      modify1D(1, 1, m);
    }
  }
  void query() {
    vmax = -INF;
    vmin = INF;
    query2D(1, 1, n);
  }
  void modify() {
    modify2D(1, 1, n);
  }
  void init(int n, int m) {
    this -> n = n;
    this -> m = m;
  }
} t;
t.x1 = max(1, x - l / 2); t.y1 = max(1, y - l / 2);
t.x2 = min(n, x + l / 2); t.y2 = min(n, y + l / 2);
t.query();
t.x = x; t.y = y; t.v = val;
t.modify();
```

2.5 SBT

```
Node *create(long long v) {
    Node *p = &memo[top++];
    p -> s = 1; p -> sum = p -> key = v;
    p -> ch[0] = p -> ch[1] = nul; return p;
}
struct SBT {
  #define L(x) ((x)->ch[0])
  #define R(x) ((x)->ch[1])
  Node *root;
  void rot(Node *&t, int f) {
    Node *k = t -> ch[f ^ 1];
    t -> ch[f ^ 1] = k -> ch[f]; k -> ch[f] = t;
    k -> s = t -> s; k -> sum = t -> sum;
    t -> s = L(t) -> s + R(t) -> s + 1;
    t -> sum = L(t) -> sum + R(t) -> sum + t -> key;
    t = k;
  }
  void maintain(Node *&t,int f) {
    if(t -> ch[f] -> ch[f] -> s > t -> ch[f ^ 1] -> s) {
      rot(t, f ^ 1);
    } else if(t -> ch[f] -> ch[f ^ 1] -> s > t -> ch[f ^ 1]
         -> s) {
      rot(t -> ch[f], f);
      rot(t, f ^ 1);
    } else return ;
    for(int i = 0; i < 2; i++) maintain(t -> ch[i], i);
    for(int i = 0; i < 2; i++) maintain(t, i);
  }
  void ins(Node *&t, long long v) {
    if(t == nul) t = create(v);
    else {
      t -> s++;
      t -> sum += v;
      ins(t -> ch[v >= t -> key], v);
      maintain(t, v >= t -> key);
    }
  }
```

```
  Node *del(Node *&t, long long v) {
    if(t == nul) return nul;
    t -> s--;
    Node *p = nul;
    if(v == t -> key || t -> ch[v > t -> key] == nul) {
      if(L(t) != nul && R(t) != nul) {
        p = del(L(t), v + 1), t -> key = p -> key;
      } else {
        p = t; t = t -> ch[L(t) == nul];
      }
      t -> sum = L(t) -> sum + R(t) -> sum + t -> key;
      return p;
    }
    p = del(t -> ch[v > t -> key], v);
    t -> sum = L(t) -> sum + R(t) -> sum + t -> key;
    return p;
  }
} sbt;
int KthL(int t, int k) { //largest
  if(t == 0 || k <= 0 || k > sbt[t].s) return 0;;
  int s = (sbt[t].ch[1] == 0? 0: sbt[sbt[t].ch[1]].s);
  if(k == s + 1) {
    return sbt[t].key;
  } else if(k <= s) {
    return KthL(sbt[t].ch[1], k);
  }
  return KthL(sbt[t].ch[0], k - s - 1);
}
int KthS(int t, int k) { //smallest
  if(t == 0 || k <= 0 || k > sbt[t].s) return 0;
  if(k <= sbt[sbt[t].ch[0]].s) {
    return KthS(sbt[t].ch[0], k);
  } else if(k > sbt[sbt[t].ch[0]].s + 1) {
    return KthS(sbt[t].ch[1], k - sbt[sbt[t].ch[0]].s - 1);
  }
  return sbt[t].key;
}
nul = &memo[top++];
nul -> ch[0] = nul -> ch[1] = nul;
nul -> s = 0;
```

2.6 Splay

```
struct Node {
  Node *ch[2], *pre;
  int s, v, lmi, rmi, lma, rma, sum;
  int rep, sw, inv;
};
char str[N];
struct Splay {
  int top;
  Node *nul, *root, memo[N];
  Node *create(int v) {
    Node* p = &memo[top++];
    p -> pre = p -> ch[0] = p -> ch[1] = nul;
    p -> s = 1; p -> v = v; p -> sum = v;
    if(v > 0) {
      p -> lmi = p -> rmi = 0; p -> lma = p -> rma = v;
    } else {
      p -> lmi = p -> rmi = v; p -> lma = p -> rma = 0;
    }
    p -> rep = p -> sw = p -> inv = 0;
    return p;/*!*/
  }
  void init() {
    top = 0;
    nul = &memo[top++];
    nul -> ch[0] = nul -> ch[1] = nul->pre = nul;
    nul -> s = 0; nul -> v = nul->sum = 0;
    nul->lmi = nul->rmi = oo; nul->lma = nul->rma = -oo;
    nul->rep = nul->sw = nul->inv = 0;
  }
  void pushup(Node* p) {
  }
  void repupd(Node* p, int v) {
    if(p == nul) {
      return;
    }
    p -> rep = v;
    p -> sw = p -> inv = 0;
    if(p -> rep > 0) {
      p -> lmi = p -> rmi = 0;
      p -> lma = p -> rma = p -> s * p -> rep;
    } else {
      p -> lma = p -> rma = 0;
      p -> lmi = p -> rmi = p -> s * p -> rep;
    }
    p -> sum = p -> s * p -> rep;
    p -> v = p -> rep;
  }
  void swupd(Node* p) {
    if(p == nul)return ;
    p -> sw ^= 1;
    swap(L(p), R(p));
    swap(p -> lmi, p -> rmi);
    swap(p -> lma, p -> rma);
  }
  void invupd(Node* p) {
    if(p == nul)return ;
    p -> inv ^= 1;
    swap(p -> lmi, p -> lma);
    p -> lmi = - p -> lmi; p -> lma = - p -> lma;
    swap(p -> rmi, p -> rma);
    p -> rmi = - p -> rmi; p -> rma = - p -> rma;
    p -> sum = - p -> sum; p -> v = -p -> v;
  }
  void pushdown(Node* p) {
    if(p -> rep) {
      repupd(L(p), p -> rep); repupd(R(p), p -> rep);
      p -> rep = 0;
    }
    if(p -> sw) {
      swupd(L(p)); swupd(R(p)); p -> sw = 0;
    }
    if(p -> inv) {
      invupd(L(p)); invupd(R(p)); p -> inv = 0;
    }
  }
  Node* build(int l, int r, Node* fa) {
    if(l > r) return nul;
    int mid = (l + r) / 2;
    Node* p;
    if(0 == str[mid]) {
      p = create(0);
    } else if('(' == str[mid]) {
      p = create(1);
    } else {
      p = create(-1);
    }
    p -> ch[0] = build(l, mid - 1, p);
    p -> ch[1] = build(mid + 1, r, p);
    p -> pre = fa;
    pushup(p);
    return p;/*!*/
  }
  void rotate(Node* x, int t) {//K>=1
    Node* y = x -> pre;
    pushdown(y); pushdown(x);
    y -> ch[t ^ 1] = x -> ch[t]; x -> pre = y -> pre;
    if(x -> ch[t] != nul)x -> ch[t] -> pre = y;
    if(y -> pre != nul) y -> pre -> ch[R(y -> pre) == y] =
        x;
    x -> ch[t] = y; y -> pre = x;
    pushup(y);/*!*/
  }
  void splay(Node* x, Node* f) {
    pushdown(x);
    while(x -> pre != f) {
      Node* y = x -> pre; Node* z = y -> pre;
      pushdown(z); pushdown(y); pushdown(x);
      if(z == f) {
        rotate(x, L(y) == x);
      } else {
        int t = (L(z) == y);
        if(y -> ch[t] == x) {
          rotate(x, t ^ 1); rotate(x, t);
        } else {
          rotate(y, t); rotate(x, t);
        }
      }
```

```
  }
  pushup(x);
  if(f == nul) {
    root = x;
  }
}
void selectk(int k, Node* f) {
  Node* p = root;
  while(1) {
    pushdown(p);
    int tmp = L(p) -> s + 1;
    if(tmp == k) {
      break;
    }
    if(tmp > k) {
      p = L(p);
    } else {
      p = R(p); k -= tmp;
    }
  }
  splay(p, f);
}
void Rep(int l, int r, int c) {
  selectk(l, nul); selectk(r + 2, root);
  repupd(L(R(root)), c);
  pushup(R(root)); pushup(root);
}
int Q(int l, int r) {
  selectk(l, nul); selectk(r + 2, root);
  int x = L(R(root)) -> lmi;
  int y = L(R(root)) -> sum;
  return (abs(y - x) + 1) / 2 + (abs(x) + 1) / 2;
}
Node *aa_bound(Node* p, int val) { //>=
  if(p == nul) return nul;
  if(p -> v < val) return aa_bound(R(p), val);
  else {
    Node* tmp = aa_bound(L(p), val);
    if(tmp == nul || p -> v < val) return p;
    return tmp;
  }
}
Node *bb_bound(Node* p, int val) { //<
  if(p == nul) return nul;
  if(p -> v >= val) return bb_bound(L(p), val);
  else {
    Node *tmp = bb_bound(R(p), val);
    if(tmp == nul || p ->v >= val) return p;
    return tmp;
  }
}
Node *cc_bound(Node* p, int val) { //>
  if(p == nul)return nul;
  if(p -> v <= val)return cc_bound(R(p), val);
  else {
    Node* tmp = cc_bound(L(p), val);
    if(tmp == nul || tmp -> v <= val) return p;
    return tmp;
  }
}
void ins(Node* q) {
  Node *bb = bb_bound(root, q -> v);
  Node *aa = aa_bound(root, q -> v);
  splay(bb, nul); splay(aa, bb);
  L(R(root)) = q; q->pre = R(root);
  pushup(R(root)); pushup(root);
}
void del(Node *p) {
  splay(p, nul);
  if(R(p) != nul) {
    root = R(p);
    selectK(1, nul);
    L(root) = L(p);
    if(L(root) != nul) {
      L(root) -> pre = root;
    }
  } else root = L(p);
  if(root != nul) {
    root->pre = nul;/*!*/
    pushup(root);
  }
  }
};
Splay tree;
tree.init();
tree.root = tree.build(0, n + 1, tree.nul);
```

2.7 Persistent Binary Tree

```
int ls[N], rs[N], c[N], tree[N], idx, mx;
int build(int l, int r) {
  int rt = idx++;
  c[rt] = 0;
  if(l != r) {
    int mid = (l + r) >> 1;
    ls[rt] = build(l, mid);
    rs[rt] = build(mid + 1, r);
  }
  return rt;
}
int update(int rt, int pos, int val) {
  int nrt = idx++, ret = nrt;
  c[nrt] = c[rt] + val;
  int l = 1, r = mx;
  while(l != r) {
    int mid = (l + r) >> 1;
    if(pos <= mid) {
      ls[nrt] = idx++; rs[nrt] = rs[rt];
      nrt = ls[nrt]; rt = ls[rt];
      r = mid;
    } else {
      rs[nrt] = idx++; ls[nrt] = ls[rt];
      nrt = rs[nrt]; rt = rs[rt];
      l = mid + 1;
    }
    c[nrt] = c[rt] + val;
  }
  return ret;
}
int query(int lrt, int rrt, int k) {
  int l = 1, r = n;
  while(l != r) {
    int mid = (l + r) >> 1;
    if(c[ls[rrt]] - c[ls[lrt]] < k) {
      k -= (c[ls[rrt]] - c[ls[lrt]]);
      lrt = rs[lrt]; rrt = rs[rrt];
      l = mid + 1;
    } else {
      lrt = ls[lrt]; rrt = ls[rrt];
      r = mid;
    }
  }
  return l;
}
sort(b + 1, b + n + 1);
n = unique(b + 1,b + n + 1) - b - 1;
tree[0] = build(1, n);
tree[i] = update(tree[i - 1], f(a[i]), 1);
query(tree[l - 1], tree[r], k)
b[query(tree[l - 1], tree[r], k)]

struct Node {/**lazy**/
  int ls, rs;
  long long lazy, sum;
  Node() {}
  Node(int ls, int rs, int lazy, int sum): ls(ls), rs(rs),
       lazy(lazy), sum(sum) {}
};
int a[100100];
Node node[N];
int tree[100100], idx, mx;
void pushup(int rt) {
  node[rt].sum = node[node[rt].ls].sum +
       node[node[rt].rs].sum;
}
int build(int l, int r) {
  int rt = idx++;
  node[rt].sum = node[rt].lazy = 0;
  if(l != r) {
    int mid = (l + r) >> 1;
```

```
    node[rt].ls = build(l, mid);
    node[rt].rs = build(mid + 1, r);
    pushup(rt);
  } else node[rt].sum = a[l];
  return rt;
}
int update(int rt, int ll, int rr, int l, int r, long long
     val) {
  int nrt = idx++;
  node[nrt] = node[rt];
  node[nrt].sum += val * (r - l + 1);
  if(ll == l && rr== r) {
    node[nrt].lazy += val;
    return nrt;
  }
  int mid = (ll + rr) >> 1;
  if(r <= mid) {
    node[nrt].ls = update(node[rt].ls, ll, mid, l, r,val);
  } else if(l > mid) {
    node[nrt].rs = update(node[rt].rs, mid + 1, rr, l, r,
         val);
  } else {
    node[nrt].ls = update(node[rt].ls, ll, mid, l, mid,
         val);
    node[nrt].rs = update(node[rt].rs, mid + 1, rr, mid +
         1, r, val);
  }
  return nrt;
}
long long query(int rt, int ll, int rr, int l, int r) {
  if(ll == l && rr == r)return node[rt].sum;
  long long ret = 0;
  ret += (r - l + 1) * node[rt].lazy;
  int mid = (ll + rr) >> 1;
  if(r <= mid) {
    ret += query(node[rt].ls, ll, mid, l, r);
  } else if(l > mid) {
    ret += query(node[rt].rs, mid + 1, rr, l, r);
  } else {
    ret += query(node[rt].ls, ll, mid, l, mid);
    ret += query(node[rt].rs, mid + 1, rr, mid + 1, r);
  }
  return ret;
}
void init(int n) {
  mx = n;
  idx = 0;
}
```

2.8 LCT

```
struct Node {
  Node *ch[2], *pre;
  int v, sz, mx[2], tot[2];
  bool isroot;
  int rep, add, sw;
};
struct LCT {
  int top;
  Node* nul, memo[N];
  Node* create(int v) {
    Node* p = &memo[top++];
    p -> pre = p -> ch[0] = p -> ch[1] = nul;
    p -> v = v; p -> sz = 1;
    p -> rep = oo; p -> add = p ->sw = 0;
    p ->isroot = 1;
    return p;/*!*/
  }
  void init() {
    top = 0;
    nul = &memo[top++];
    nul -> ch[0] = nul -> ch[1] = nul -> pre = nul;
    nul ->v = oo; nul ->sz = 0;
    nul ->rep = oo; nul ->add = nul ->sw = 0;
    nul ->isroot = 0;
  }
  void pushup(Node* p) {
    p -> sz = L(p) -> sz + R(p) -> sz + 1;
  }
  void repupd(Node* p, int v) {
    if(p == nul)return ;
    p ->v = v; p ->rep = v; p ->add = 0;
  }
  void addupd(Node* p, int add) {
    if(p == nul)return ;
    p -> v += add; p -> add += add;
  }
  void swupd(Node* p) {
    if(p == nul)return ;
    swap(L(p), R(p)); p -> sw ^= 1;
  }
  void pushdown(Node* p) {
    if(p -> rep != oo) {
      repupd(L(p), p -> rep);
      repupd(R(p), p -> rep);
      p -> rep = oo;
    }
    if(p -> sw) {
      swupd(L(p)); swupd(R(p));
      p -> sw = 0;
    }
    if(p -> add) {
      addupd(L(p), p -> add);
      addupd(R(p), p -> add);
      p -> add = 0;
    }
  }
  void down(Node* p) {
    if(!p -> isroot)down(p -> pre);
    pushdown(p);
  }
  void rot(Node* x) {
    Node *y = x -> pre, *z = y -> pre;
    int t = (x == y -> ch[0]);
    y -> ch[!t] = x -> ch[t];
    x -> ch[t] -> pre = y; x -> ch[t] = y;
    y -> pre = x; x -> pre = z;
    if(y -> isroot) {
      y -> isroot = 0; x -> isroot = 1;
    } else {
      z -> ch[y == z -> ch[1]] = x;
    }
    pushup(y);
  }
  void splay(Node* x) {
    down(x);
    while(!x->isroot) {
      Node* y = x -> pre, *z = y -> pre;
      if(!y -> isroot) {
        Node* tmp = ((x == y -> ch[0]) == (y == z ->
             ch[0])? y: x);
        rot(tmp);/*!*/
      }
      rot(x);
    }
    pushup(x);
  }
  void access(Node* p) {
    for(Node* q = nul; p != nul; p = p -> pre) {
      splay(p);
      p -> ch[1] -> isroot = 1; q -> isroot = 0;
      p -> ch[1] = q; q = p;
      pushup(p);
    }
  }
  void change_root(Node* p) {
    access(p); splay(p); swupd(p);
  }
  void cut(Node* a, Node* b) {
    change_root(a); access(a); splay(b);
    b->pre = nul;
  }
  void link(Node* a, Node* b) {
    change_root(b);
    b->pre = a;
  }
  void gao1(Node* a, Node* b, Node* c, Node* d) {
    cut(a, b); link(c, d);
  }
  void gao2(Node* a, Node* b, int c) {
```

```
    change_root(a); access(b); splay(b);
    repupd(b, c);
  }
  bool judge(Node* p, Node* q) {
    while(p -> pre != nul)p = p -> pre;
    while(q -> pre != nul)q = q -> pre;
    return p == q;
  }
};
for(int i = 1; i <= n; i++) {
  int x;
  scanf("%d", &x);
  pos[i] = tree.create(x);
}
for(int i = 1; i < n; i++) {
  int x, y;
  scanf("%d%d", &x, &y);
  tree.link(pos[x], pos[y]);
}
lct.change_root(re[x]); lct.access(re[x]);
re[x]->val=y; lct.pushup(re[x]);//w[x]=y;

void access(Node *p) {
  for(Node *q=nul; p!=nul; p=p->pre) {
    splay(p);
    (p->s[0]+=R(p)->s[1])%=mod;
    (p->s[0]+=mod-q->s[1])%=mod;
    (p->s[2]+=R(p)->s[1]*R(p)->s[1]%mod)%=mod;
    (p->s[2]+=mod-q->s[1]*q->s[1]%mod)%=mod;
    R(p)->isroot=1;
    q->isroot=0;
    p->ch[1]=q;
    q=p;
    pushup(p);
  }
}
void link(Node *a,Node *b) {
  change_root(b);
  (a->s[0]+=b->s[1])%=mod;
  (a->s[1]+=b->s[1])%=mod;
  (a->s[2]+=b->s[1]*b->s[1]%mod)%=mod;
  (a->s[3]+=b->s[1]*b->s[1]%mod)%=mod;
  b->pre=a;
}
for(int i=1; i<=n; i++)
  for(int j=0; j<g[i].size(); j++)
    lct.link(re[i],re[g[i][j]]);
```

2.9 Union Set

```
int f(int x) {
  int o = x ;
  while ( p[o] != o ) o = p[o] ;
  int ans = o ;
  while ( p[x] != x ) {
    int tmp = p[x] ; p[x] = tmp ; x = tmp ;
  }
  return ans ;
}
void Union(int uu, int vv, bool in) {
  int u = f(uu), v = f(vv);
  if(u == v)return;
  if(in) {
    st[++top] = Re(u, v, cnt[u], cnt[v], rk[u], rk[v]);
    ret += 1LL * cnt[u] * cnt[v];
  }
  if(rk[u] <= rk[v]) {
    rk[v] = max(rk[v], rk[u] + 1);
    p[u] = v;
    cnt[v] += cnt[u];
  } else {
    rk[u] = max(rk[u], rk[v] + 1);
    p[v] = u;
    cnt[u] += cnt[v];
  }
}
```

2.10 Cdq

```
struct Re {
  int x, y, z, val, idx;
  Re() {}
  Re(int x, int y, int z, int val, int idx) : x(x), y(y),
      z(z), val(val), idx(idx) {}
} re[N];
bool cmpx(const Re& a, const Re& b) {
  if(a.x ^ b.x)return a.x < b.x;
  return a.idx < b.idx;
}
bool cmpy(const Re& a, const Re& b) {
  if(a.y ^ b.y)return a.y < b.y;
  return a.idx < b.idx;
}
void upd(int l, int r) {
  if(l == r)return ;
  int mid = (l + r) >> 1;
  upd(l, mid);
  upd(mid + 1, r);
  aa = 0, bb = 0;
  for(int i = l; i <= mid; i++) {
    mema[aa++] = pool[i];
  }
  for(int i = mid + 1; i <= r; i++) {
    memb[bb++] = pool[i];
  }
  sort(mema, mema + aa, cmpy);
  sort(memb, memb + bb, cmpy);
  for(int i = 0, j = 0; j < bb; j++) {
    for(; i < aa && mema[i].y <= memb[j].y; i++) {
      if(!mema[i].val)update(mema[i].z, 1);
    }
    if(memb[j].val)ans[memb[j].idx] += memb[j].val *
        query(memb[j].z);
  }
  for(int i = 0; i < aa; i++)clc(mema[i].z);
}
void solve(int l, int r) {
  if(l == r)return ;
  int mid = (l + r) >> 1;
  solve(l, mid);
  solve(mid + 1, r);
  tt = 0;
  for(int i = l; i <= mid; i++) {
    if(!re[i].val)pool[tt++] = re[i];
  }
  for(int i = mid + 1; i <= r; i++) {
    if(re[i].val)pool[tt++] = re[i];
  }
  sort(pool, pool + tt, cmpx);
  if(tt)upd(0, tt - 1);
}
```

2.11 Binary Search

```
void solve(int head, int tail, int l, int r) {
  if (head > tail) return;
  if (l == r) {
    for (int i = head; i <= tail; ++i) {
      if (q[i].qt == 3) ans[q[i].index] = l;
    }
    return;
  }
  int mid = (l + r) >> 1;
  for (int i = head; i <= tail; ++i) {
    if (q[i].qt == 1 && q[i].y <= mid) {
      add(q[i].x, 1);
    } else if (q[i].qt == 2 && q[i].y <= mid) {
      add(q[i].x, -1);
    } else {
      tmp[i] = query(q[i].y) - query(q[i].x - 1);
    }
  }
  for (int i = head; i <= tail; ++i) {
    if (q[i].qt == 1 && q[i].y <= mid) {
      add(q[i].x, -1);
```

```
    } else if (q[i].qt == 2 && q[i].y <= mid) {
      add(q[i].x, 1);
    }
  }
  int l1=0, l2 = 0;
  for (int i = head; i <= tail; ++i) {
    if (q[i].qt == 3) {
      if (q[i].cur + tmp[i] >= q[i].k) {
        q1[++l1] = q[i];
      } else {
        q[i].cur += tmp[i];
        q2[++l2] = q[i];
      }
    } else {
      if (q[i].y <= mid) q1[++l1] = q[i];
      else q2[++l2] = q[i];
    }
  }

  for (int i = 1; i <= l1; ++i) {
    q[head + i - 1] = q1[i];
  }
  for (int i = 1; i <= l2; ++i) {
    q[head + l1 + i - 1] = q2[i];
  }
  solve(head, head + l1 - 1, l, mid);
  solve(head + l1, tail, mid + 1, r);
}
int main() {
  while (scanf("%d",&n)!=EOF) {
    qtop = 0;
    tot = 0;
    for (int i = 1; i <= n; ++i) {
      scanf("%d", a + i);
      b[tot++] = a[i];
      q[++qtop].qt = 1;
      q[qtop].x = i;
      q[qtop].y = a[i];
    }
    int cmd;
    int xi, yi,ki;
    ansid = 0;
    scanf("%d",&m);
    for (int i = 0; i < m; ++i) {
      scanf("%d", &cmd);
      if (2==cmd) {
        scanf("%d%d%d", &xi, &yi, &ki);
        q[++qtop].x = xi; q[qtop].y = yi;
        q[qtop].k = ki; q[qtop].qt = 3;
        q[qtop].cur = 0;
        q[qtop].index = ++ansid;
      } else {
        scanf("%d%d", &xi, &yi);
        q[++qtop].x = xi; q[qtop].y = a[xi];
        q[qtop].qt = 2;
                q[++qtop].x = xi; q[qtop].y = yi;
        q[qtop].qt = 1;
        a[xi] = yi;
        b[tot++] = yi;
      }
    }
    sort(b, b + tot);
    bsize = unique(b, b + tot) - b;
    for (int i = 1; i <= qtop; ++i) {
      if (q[i].qt == 1 || q[i].qt == 2) {
        q[i].y = lower_bound(b, b + bsize, q[i].y) - b + 1;
      }
    }
    solve(1, qtop, 1, bsize);
    for (int i = 1; i <= ansid; ++i) {
      printf("%d\n", b[ans[i]-1]);
    }
  }
  return 0;
}
```

2.12 Dominator Tree

```
const int vector_num = 50000;
vector<int> succ[vector_num + 10], prod[vector_num + 10],
      bucket[vector_num + 10], dom_t[vector_num + 10];
int semi[vector_num + 10], anc[vector_num + 10],
      idom[vector_num + 10], best[vector_num + 10],
      fa[vector_num + 10];
int dfn[vector_num + 10], redfn[vector_num + 10];
int child[vector_num + 10], size[vector_num + 10];
int timestamp;
void dfs(int now) {
  dfn[now] = ++timestamp;
  redfn[timestamp] = now;
  anc[timestamp] = idom[timestamp] = child[timestamp] =
        size[timestamp] = 0;
  semi[timestamp] = best[timestamp] = timestamp;
  int sz = succ[now].size();
  for(int i = 0; i < sz; ++i) {
    if(dfn[succ[now][i]] == -1) {
      dfs(succ[now][i]);
      fa[dfn[succ[now][i]]] = dfn[now];
    }
    prod[dfn[succ[now][i]]].push_back(dfn[now]);
  }
}
void compress(int now) {
  if(anc[anc[now]] != 0) {
    compress(anc[now]);
    if(semi[best[now]] > semi[best[anc[now]]])
      best[now] = best[anc[now]];
    anc[now] = anc[anc[now]];
  }
}
inline int eval(int now) {
  if(anc[now] == 0)
    return now;
  else {
    compress(now);
    return semi[best[anc[now]]] >= semi[best[now]] ?
          best[now]
           : best[anc[now]];
  }
}
inline void link(int v, int w) {
  int s = w;
  while(semi[best[w]] < semi[best[child[w]]]) {
    if(size[s] + size[child[child[s]]] >= 2*size[child[s]])
        {
      anc[child[s]] = s;
      child[s] = child[child[s]];
    } else {
      size[child[s]] = size[s];
      s = anc[s] = child[s];
    }
  }
  best[s] = best[w];
  size[v] += size[w];
  if(size[v] < 2*size[w])
    swap(s, child[v]);
  while(s != 0) {
    anc[s] = v;
    s = child[s];
  }
}
void lengauer_tarjan(int n) { // n is the vertices' number
  memset(dfn, -1, sizeof dfn);
  memset(fa, -1, sizeof fa);
  timestamp = 0;
  dfs(n);
  fa[1] = 0;
  for(int w = timestamp; w > 1; --w) {
    int sz = prod[w].size();
    for(int i = 0; i < sz; ++i) {
      int u = eval(prod[w][i]);
      if(semi[w] > semi[u])
        semi[w] = semi[u];
    }
    bucket[semi[w]].push_back(w);
    //anc[w] = fa[w]; link operation for o(mlogm) version
    link(fa[w], w);
    if(fa[w] == 0)
      continue;
    sz = bucket[fa[w]].size();
```

```
    for(int i = 0; i < sz; ++i) {
      int u = eval(bucket[fa[w]][i]);
      if(semi[u] < fa[w])
        idom[bucket[fa[w]][i]] = u;
      else
        idom[bucket[fa[w]][i]] = fa[w];
    }
    bucket[fa[w]].clear();
  }
  for(int w = 2; w <= timestamp; ++w) {
    if(idom[w] != semi[w])
      idom[w] = idom[idom[w]];
  }
  idom[1] = 0;
  for(int i = timestamp; i > 1; --i) {
    if(fa[i] == -1)
      continue;
    dom_t[idom[i]].push_back(i);
  }
}
long long ans[50010];
void get_ans(int now) {
  ans[redfn[now]] += redfn[now];
  int sz = dom_t[now].size();
  for(int i = 0; i < sz; ++i) {
    ans[redfn[dom_t[now][i]]] += ans[redfn[now]];
    get_ans(dom_t[now][i]);
  }
}
void init(int n, int m) {
  for(int i=0; i<=n; i++)
    succ[i].clear(), prod[i].clear(), bucket[i].clear(),
        dom_t[i].clear();
  memset(ans,0,sizeof(*ans)*(n+3));
}
while(scanf("%d%d",&n,&m)!=EOF) {
  init(n,m);
  for(int i=0,u,v; i<m; i++) {
    scanf("%d%d",&u,&v);
    succ[u].push_back(v);
  }
  lengauer_tarjan(n);
  get_ans(1);
  for(int i=1; i<=n; i++)
    printf("%I64d%c", ans[i], i == n ? '\n' : ' ');
}
```

3 DP

3.1 2D-LIS

```
set<pair<int,int> > re[100100];
int cc;
const int oo = (1 << 30);
bool judge(int idx, pair<int, int> x) {
  if(re[idx].empty())return true;
  set<pair<int,int> >::iterator it = re[idx].lower_bound(
      {x.first - 1, oo});
  if(it == re[idx].begin())return false;
  it--;
  if((*it).first < x.first && (*it).second <
      x.second)return true;
  return false;
}
void update(pair<int, int> x) {
  if(0 == cc || judge(cc - 1, x)) {
    re[cc++].insert(x);
    return ;
  }
  int l = -1,r = cc - 1;
  while(l + 1 < r) {
    int mid = (l + r) >> 1;
    if(judge(mid, x)) {
      l = mid;
    } else {
      r = mid;
    }
  }
  set<pair<int,int> >::iterator it = re[r].lower_bound(x);
  if(it != re[r].begin()) {
    it--;
    if((*it).second <= x.second)return ;
    it++;
  }
  while(it != re[r].end() && x.second <=
      (*it).second)re[r].erase(it++);
  re[r].insert(x);
}
```

3.2 2D-RMQ

```
void init() {
  for(int i = 0; i < line; i++)
    for(j = 0; j < row; j++)
      dp[i][j][0][0] = mat[i][j];
  for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
      if(i == 0 && j == 0)continue;
      for(int l = 0; l + C(i) <= line; l++) {
        for(int r = 0; r + C(j) <= row; r++) {
          if(i == 0)dp[l][r][i][j] = max(dp[l][r][i][j - 1],
              dp[l][r + C(j - 1)][i][j - 1]);
          else dp[l][r][i][j] = max(dp[l][r][i - 1][j],dp[l
              + C(i - 1)][r][i - 1][j]);
        }
      }
    }
  }
}
int Q(int x1, int y1, int x2, int y2) {
  int lx = Log[x2 - x1 + 1];
  int ly = Log[y2 - y1 + 1];
  int ret = max(dp[x1][y1][lx][ly], dp[x2 - C(lx) +
      1][y1][lx][ly]);
  return max(ret, max(dp[x1][y2 - C(ly) + 1][lx][ly], dp[x2
      - C(lx) + 1][y2 - C(ly) + 1][lx][ly]));
}
```

3.3 LCA

```
void update(int u, int fa) {
  dep[u] = dep[fa] + 1;
  dp[u][0] = fa;
  for(int i = 1; i < 17; i++) {
    dp[u][i] = dp[dp[u][i - 1]][i - 1];
  }
}
int lca(int u, int v) {
  if(dep[u] < dep[v]) swap(u, v);
  for(int df = dep[u] - dep[v], t = 0; df; df >>= 1, t++)
    if(df & 1) u = dp[u][t];
  if(u == v) return u;
  for(int i = 16; i >= 0; i--) {
    if(dp[u][i] != dp[v][i]) {
      u = dp[u][i];
      v = dp[v][i];
    }
  }
  return dp[u][0];
}

int F[MAXN*2];//Euler Sequence
int P[MAXN];//first time point i appears in F
void dfs(int u,int pre,int dep) {
  F[++cnt] = u;
  rmq[cnt] = dep;
  P[u] = cnt;
  for(int i = head[u]; i != -1; i = edge[i].next) {
    int v = edge[i].to;
    if(v == pre)continue;
    dfs(v,u,dep+1);
    F[++cnt] = u;
    rmq[cnt] = dep;
  }
```

```
}
dp[i][j] = rmq[dp[i][j-1]] <
      rmq[dp[i+(1<<(j-1))][j-1]]?dp[i][j-1]:dp[i+(1<<(j-1))][j-1];
return rmq[dp[a][k]] <=
      rmq[dp[b-(1<<k)+1][k]]?dp[a][k]:dp[b-(1<<k)+1][k];
return F[st.query(P[u],P[v])];
```

3.4 Steiner Tree

```
for (int i = 1; i <= n; ++i)
  for (int j = 0; j < 1 << 4; ++j)
    if (dis[i][j] == 0) que.push((Opt) {0, i, j});
while (!que.empty()) {
  int x = que.top().x, mask = que.top().mask;
  que.pop();
  if (bo[x][mask]) continue;
  bo[x][mask] = 1;
  for (int k = first[x]; k != -1; k = edge[k].nex) {
    int y = edge[k].y;
    if (dis[y][mask] > dis[x][mask] + edge[k].s) {
      dis[y][mask] = dis[x][mask] + edge[k].s;
      que.push((Opt) {dis[y][mask], y, mask});
    }
  }
  for (int mask1 = 0; mask1 < 1 << 4; ++mask1)
    if (dis[x][mask | mask1] > dis[x][mask] +
          dis[x][mask1]) {
      dis[x][mask | mask1] = dis[x][mask] + dis[x][mask1];
      que.push((Opt) {dis[x][mask | mask1], x, mask |
            mask1});
    }
}
```

4 Graph

4.1 KM

```
const int INF = 1000000000;
int n;
int w[N][N], u[N], v[N], p[N], minv[N], fa[N];
bool used[N];
void km(int lev) {
  int i = lev;
  for (int j = 0; j <= n; ++j) {
    minv[j] = INF; used[j] = false;
  }
  p[n] = i;
  int j0 = n;
  do {
    used[j0] = true;
    int i0 = p[j0], delta = INF, j1;
    for (int j = 0; j < n; ++j) {
      if (!used[j]) {
        int cur = w[i0][j] - u[i0] - v[j];
        if (cur < minv[j]) {
          minv[j] = cur; fa[j] = j0;
        }
        if (minv[j] < delta) {
          delta = minv[j]; j1 = j;
        }
      }
    }
    for (int j = 0; j <= n; ++j) {
      if (used[j]) {
        u[p[j]] += delta, v[j] -= delta;
      } else {
        minv[j] -= delta;
      }
    }
    j0 = j1;
  } while (p[j0] != -1);
  do {
    int j1 = fa[j0]; p[j0] = p[j1]; j0 = j1;
  } while (j0 != n);
}
void solve() {
  for(int i = 0; i < n; i++) {
    km(i);
  }
  printf("%d\n", v[n]);
}
mst(p, -1); mst(u, 0); mst(v, 0);
for(int i = 0; i < n; i++) {
  for(int j = 0; j < n; j++) {
    scanf("%d", &w[i][j]);
    w[i][j] = -w[i][j];
  }
}
solve();
```

4.2 Hungary

```
int vis[N * N], match[N * N];
int fi(int u) {
  for(int temp = head[u]; temp != -1; temp =
        edge[temp].next) {
    int v = edge[temp].to;
    if(vis[v])continue;
    vis[v] = true;
    if(-1 == match[v] || fi(match[v])) {
      match[v] = u;
      return true;
    }
  }
  return false;
}
int Hungary() {
  int ret = 0;
  memset(match, -1, sizeof(match));
  for(int i = 0; i < n * m; i++) {
    memset(vis, 0, sizeof(vis));
    if(fi(i))ret++;
  }
  return ret;
}
```

4.3 HK

```
int dist[N << 1], mx[N], my[N], m, n;
vector<int> map[N];
int que[N << 1], head, tail;
int bfs() {
  int i;
  head = 0; tail = -1;
  for(i = 1; i <= n; i++)
    if(mx[i] == -1) que[++tail] = i;
  for(i = 0; i <= m + n; i++) dist[i] = 0;
  int flag = 0;
  while(head <= tail) {
    int u = que[head++];
    for(i = 0; i < map[u].size(); i++) {
      int v = map[u][i];
      if(dist[n + v] == 0) {
        dist[n + v] = dist[u] + 1;
        if(my[v] != -1) {
          dist[my[v]] = dist[n + v] + 1;
          que[++tail] = my[v];
        } else flag = 1;
      }
    }
  }
  return flag;
}
int dfs(int u) {
  for(int i = 0; i < map[u].size(); i++) {
    int v = map[u][i];
    if(dist[u] + 1 == dist[v + n]) {
      int t = my[v]; dist[v + n] = 0;
      if(t == -1 || dfs(t)) {
        my[v] = u; mx[u] = v;
        return 1;
      }
    }
```

```
  }
  return 0;
}
int H_K() {
  int i;
  for(i = 0; i <= n; i++) mx[i] = -1;
  for(i = 0; i <= m; i++) my[i] = -1;
  int ans = 0;
  while(bfs()) {
    for(i = 1; i <= n; i++)
      if(mx[i] == -1 && dfs(i)) ans++;
  }
  return ans;
}
```

4.4 2-SAT

```
bool Dfs(int u) {
  if(mark[u ^ 1])return false;
  if(mark[u])return true;
  mark[u] = true;
  stack[++top] = u;
  int temp;
  for(temp = head[u]; temp != -1; temp = edge[temp].next)
    if(!Dfs(edge[temp].to))return false;
  return true;
}
bool Solve() {
  memset(mark, 0, sizeof(mark));
  for(int i = 0; i < 2 * n; i += 2) {
    if(!mark[i] && !mark[i+1]) {
      top = 0;
      if(!Dfs(i)) {
        while(top)mark[stack[top--]] = false;
        if(!Dfs(i + 1))return false;
      }
    }
  }
  return true;
}
```

4.5 SAP

```
struct F {
  void init(int N) {
    this->N = N;
    mst(head, -1); countedge = 0;
  }
  //Edge(head[s], t, cap);Edge(head[t], s, 0);
  int sap(const int& s, const int& t) {
    mst(numh, 0); mst(h, 0); mst(pre, -1);
    int i;
    for(i = 0; i < N; i++) curedge[i] = head[i];
    numh[0] = N;
    int u = s, curflow, neck, ret = 0;
    while(h[s] < N) {
      if(u == t) {
        curflow = oo;
        for(i = s; i != t; i = edge[curedge[i]].to) {
          if(curflow > edge[curedge[i]].cap) {
            curflow = edge[curedge[i]].cap;
            neck = i;
          }
        }
        for(int i = s; i != t; i = edge[curedge[i]].to) {
          int e = curedge[i];
          edge[e].cap -= curflow;
          edge[e ^ 1].cap += curflow;
        }
        ret += curflow;
        u = neck;
      }
      for(i = curedge[u]; i != -1; i = edge[i].next) {
        if(edge[i].cap && h[edge[i].to] + 1 == h[u]) break;
      }
      if(i != -1) {
        curedge[u] = i;
        pre[edge[i].to] = u;
        u = edge[i].to;
      } else {
        if(0 == --numh[h[u]])break;
        curedge[u] = head[u];
        int H = N;
        for(i = head[u]; i != -1; i = edge[i].next) {
          if(edge[i].cap)H = min(H, h[edge[i].to] + 1);
        }
        h[u] = H;
        numh[H]++;
        if(u != s) u = pre[u];
      }
    }
    return ret;
  }
  bool s_side[MAXN];
  void bfs(const int& s) {
    mst(s_side, 0);
    queue<int> que; que.push(s);
    while(!que.empty()) {
      int u = que.front();
      que.pop();
      for(int tmp = head[u]; tmp != -1; tmp =
          edge[tmp].next) {
        int v = edge[tmp].to;
        if(edge[tmp].cap && !s_side[v]) {
          s_side[v] = true;
          que.push(v);
        }
      }
    }
  }
  int p[MAXN], c[MAXN][MAXN];
  void gusfield() {
    mst(p, 0); mst(c, 63);
    for(int i = 1; i < N; i++) {
      for(int j = 0; j < countedge; j += 2) {
        edge[j].cap += edge[j ^ 1].cap;
        edge[j ^ 1].cap = 0;
      }
      int f = sap(i, p[i]);
      bfs(i);
      for(int j = i + 1; j < N; j++) {
        if(s_side[j] && p[j] == p[i]) {
          p[j] = i;
        }
      }
      c[i][p[i]] = c[p[i]][i] = f;
      for(int j = 0; j < i; j++) {
        c[i][j] = c[j][i] = min(f, c[p[i]][j]);
      }
    }
    for(int i = 0; i < N; i++)
      c[i][i] = 0;
  }
} f;

mst(h, -1);
h[t] = 0; numh[0] = 1;
queue<int> que; que.push(t);
while(!que.empty()) {
  int u = que.front(); que.pop();
  for(int tmp = head[u]; tmp != -1; tmp = edge[tmp].next) {
    int v = edge[tmp].to;
    if(h[v] != -1) continue;
    que.push(v); h[v] = h[u] + 1; ++numh[h[v]];
  }
}

c(u,v) = 1; c(s,v) = oo; c(v,t) = oo + 2 * g - du(v)
c(u,v) = we; c(s,v) = oo; c(v,t) = oo + 2 * g -
     sumofedge(v) - 2 * p(v)
h(g) = (m * n - cut(S,T)) / 2

for(e: E){
       cap(u, v) = up(u, v) - low(u, v)
       cap(S', v) = low(u, v)
       cap(v, T') = low(u, v)
       tflow += low(u, v)
}
```

```
int ans1 = f.sap(S', T');
cap(T, S) = oo
int ans2 = f.sap(S', T');
if(ans1 + ans2 == sum) ans2;
else puts("Impossible");
```

4.6 MCMF

```
struct MCMF {
  Edge edge[MAXM];
  int head[MAXN], countedge;
  void Init(int N) {
    this->N = N;
    memset(head, -1, sizeof(head));
    countedge = 0;
  }
  int N;
  int inq[MAXN], dis[MAXN], pre[MAXN], ad[MAXN];
  bool BF(int s, int t, int& cost) {
    int i;
    for(i = 0; i < N; i++)dis[i] = INF;
    memset(inq, 0, sizeof(inq));
    dis[s] = 0; inq[s] = 1; pre[s] = -1; ad[s] = INF;
    queue<int> que; que.push(s);
    int u, v, temp;
    while(!que.empty()) {
      u = que.front(); que.pop();
      inq[u] = 0;
      for(temp = head[u]; temp != -1; temp =
            edge[temp].next) {
        v = edge[temp].to;
        if(edge[temp].cap && dis[v] > dis[u] +
              edge[temp].cost) {
          dis[v] = dis[u] + edge[temp].cost; pre[v] = temp;
          ad[v] = min(ad[u], edge[temp].cap);
          if(!inq[v]) {
            inq[v] = 1; que.push(v);
          }
        }
      }
    }
    if(dis[t] == INF)return false;
    cost += dis[t] * ad[t];
    u = t;
    while(u != s) {
      edge[pre[u]].cap -= ad[t]; edge[pre[u] ^ 1].cap +=
            ad[t];
      u = edge[pre[u]].from;
    }
    return true;
  }

  int MinC(int s, int t) {
    int cost = 0;
    while(BF(s, t, cost));
    return cost;
  }
};

/*****zkw*****/
struct MaxFlow {
  int N;
  int st, en, maxflow, mincost;
  bool vis[MAXN];
  Edge edge[MAXM];
  int head[MAXN], countedge, pre[MAXN], cur[MAXN],
      dis[MAXN];
  std::queue <int> Q;
  void init(int N) {
    this->N = N;
    memset(head, -1, sizeof(head));
    countedge = 0;
  }
  bool modell() {
    int v, min = oo;
    for(int i = 0; i <= N; i++) {
      if(!vis[i])
        continue;
      for(int j = head[i]; v = edge[j].v, j != -1; j =
            edge[j].next)
        if(edge[j].cap)
          if(!vis[v] && dis[v]-dis[i]+edge[j].cost < min)
            min = dis[v] - dis[i] + edge[j].cost;
    }
    if(min == oo)
      return false;
    for(int i = 0; i <= N; i++)
      if(vis[i])
        cur[i] = head[i], vis[i] = false, dis[i] += min;
    return true;
  }
  int augment(int i, int flow) {
    if(i == en) {
      mincost += dis[st] * flow; maxflow += flow;
      return flow;
    }
    vis[i] = true;
    for(int j = cur[i], v; v = edge[j].v, j != -1; j =
          edge[j].next) {
      if(!edge[j].cap)
        continue;
      if(vis[v] || dis[v]+edge[j].cost != dis[i])
        continue;
      int delta = augment(v, min(flow, edge[j].cap));
      if(delta) {
        edge[j].cap -= delta; edge[j^1].cap += delta;
        cur[i] = j;
        return delta;
      }
    }
    return 0;
  }
  void spfa() {
    int u, v;
    for(int i = 0; i <= N; i++)
      vis[i] = false, dis[i] = oo;
    dis[st] = 0; Q.push(st); vis[st] = true;
    while(!Q.empty()) {
      u = Q.front(), Q.pop(); vis[u] = false;
      for(int i = head[u]; v = edge[i].v, i != -1; i =
            edge[i].next) {
        if(!edge[i].cap || dis[v] <= dis[u] + edge[i].cost)
          continue;
        dis[v] = dis[u] + edge[i].cost;
        if(!vis[v]) {
          vis[v] = true; Q.push(v);
        }
      }
    }
    for(int i = 0; i <= N; i++)
      dis[i] = dis[en] - dis[i];
  }
  int zkw(int s, int t) {
    st = s, en = t;
    spfa();
    mincost = maxflow = 0;
    for(int i = 0; i <= N; i++)
      vis[i] = false, cur[i] = head[i];
    do {
      while(augment(st, oo))
        memset(vis, false, sizeof(vis));
    } while(modell());
    return mincost;
  }
} f;
```

4.7 BCC

```
void dfs(int u, int fa) {
  dfn[u] = low[u] = ++idx;
  int child = 0;
  for(int temp = head[u]; temp != -1; temp =
        edge[temp].next) {
    int v = edge[temp].to;
    pair<int,int> e = make_pair(u, v);
    if(!dfn[v]) {
      st.push(e);
      child++;
```

```
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if(low[v] >= dfn[u]) {
        iscut[u] = true;
        bcc_cnt++;
        while(1) {
          pair<int,int> x = st.top();
          st.pop();
          if(bccno[x.first] != bcc_cnt)
            bccno[x.first] = bcc_cnt;
          if(bccno[x.second] != bcc_cnt)
            bccno[x.second] = bcc_cnt;
          if(x.first == u && x.second == v)break;
        }
      }
    } else if(dfn[v] < dfn[u] && v != fa) {
      st.push(e);
      low[u] = min(low[u], dfn[v]);
    }
  }
  if(fa < 0 && child == 1) {
    iscut[u] = 0;
  }
}
void find_bcc(int n) {
  pre = 0;
  mst(dfn, 0);mst(iscut, 0); mst(bccno, 0);
  idx = bcc_cnt = 0;
  while(!st.empty())st.pop();
  for(int i = 0; i < n; i++) {
    if(!dfn[i]) {
      dfs(i, -1);
      pre++;
    }
  }
}
```

4.8 SCC

```
int dfn[N], low[N], sccno[N], scc_cnt, idx;
stack<int> st;
void dfs(int u) {
  dfn[u] = low[u] = ++idx;
  st.push(u);
  for(int temp = head[u]; temp != -1; temp =
       edge[temp].next) {
    int v = edge[temp].to;
    if(!dfn[v]) {
      dfs(v);
      low[u] = min(low[u], low[v]);
    } else if(!sccno[v]) {
      low[u] = min(low[u], dfn[v]);
    }
  }
  if(low[u] == dfn[u]) {
    scc_cnt++;
    while(1) {
      int x = st.top();
      st.pop();
      sccno[x] = scc_cnt;
      W[scc_cnt] += w[x];
      if(x == u) {
        break;
      }
    }
  }
}
void find_scc(int n) {
  idx = scc_cnt = 0;
  while(!st.empty()) {
    st.pop();
  }
  memset(sccno, 0, sizeof(sccno));
  memset(dfn, 0, sizeof(dfn));
  for(int i = 1; i <= n; i++) {
    if(!dfn[i]) {
      dfs(i);
    }
  }
}
```

4.9 DLX

```
struct DLX {
  int U[M], D[M], L[M], R[M], C[M];
  int H[N], S[N], Q[N];
  int size, ret;
  void remove(const int& c) {
    L[R[c]] = L[c]; R[L[c]] = R[c];
    for(int i = D[c]; i != c; i = D[i]) {
      for(int j = R[i]; j != i; j = R[j]) {
        U[D[j]] = U[j]; D[U[j]] = D[j]; --S[C[j]];
      }
    }
  }
  void resume(const int& c) {
    for(int i = U[c]; i != c; i = U[i]) {
      for(int j = L[i]; j != i; j = L[j]) {
        ++S[C[j]]; U[D[j]] = j; D[U[j]] = j;
      }
    }
    L[R[c]] = c; R[L[c]] = c;
  }
  void dfs(const int& k) {
    if(k>=ret) {
      return ;
    }
    if(!R[0]) {
      ret = k;
      return ;
    }
    int c;
    for(int tmp = M, i = R[0]; i; i = R[i]) {
      if(S[i] < tmp) {
        tmp = S[c = i];
      }
    }
    remove(c);
    for(int i = D[c]; i != c; i = D[i]) {
      for(int j = R[i]; j != i; j = R[j]) {
        remove(C[j]);
      }
      dfs(k + 1);
      for(int j = L[i]; j != i; j = L[j]) {
        resume(C[j]);
      }
    }
    resume(c);
  }
  void update(const int& r,const int& c) {
    ++S[C[size] = c];
    D[size] = D[c]; U[D[c]] = size;
    U[size] = c; D[c] = size;
    if(H[r] < 0)H[r] = L[size] = R[size] = size;
    else {
      R[size] = R[H[r]]; L[R[H[r]]] = size;
      L[size] = H[r]; R[H[r]] = size;
    }
    size++;
  }
  void init(int col) {
    ret = (1 << 30);
    for(int i = 0; i <= col; i++) {
      S[i] = 0; D[i] = U[i] = i;
      L[i + 1] = i; R[i] = i + 1;
    }
    R[col] = 0; size = col + 1;
    memset(H, -1, sizeof(H));
  }
}dlx;

struct DLX {
  int U[M], D[M], L[M], R[M], C[M], X[M];
  int H[N], S[N], Q[N];
  int size;
  void remove(const int& c) {
    for(int i = D[c]; i != c; i = D[i]) {
      R[L[i]] = R[i]; L[R[i]] = L[i];
```

```
    }
  }
  void resume(const int& c) {
    for(int i = U[c]; i != c; i = U[i]) {
      R[L[i]] = i; L[R[i]] = i;
    }
  }
  int h() {
    int ret = 0;
    bool vis[60];
    memset (vis, false, sizeof(vis));
    for (int i = R[0]; i; i = R[i]) {
      if(vis[i])continue;
      ret++;
      vis[i] = true;
      for (int j = D[i]; j != i; j = D[j])
        for (int k = R[j]; k != j; k = R[k])
          vis[C[k]] = true;
    }
    return ret;
  }
  int ret;
  void dfs(const int& k) {
    if(!R[0] || k + h() >= ret) {
      if(!R[0]) ret = min(ret, k);
      return ;
    }
    int c;
    for(int tmp = M, i = R[0]; i; i = R[i]) {
      if(S[i] < tmp) {
        tmp = S[c = i];
      }
    }
    for(int i = D[c]; i != c; i = D[i]) {
      Q[k] = i;
      remove(i);
      for(int j = R[i]; j != i; j = R[j]) {
        remove(j);
      }
      dfs(k + 1);
      for(int j = L[i]; j != i; j = L[j]) {
        resume(j);
      }
      resume(i);
    }
  }
  void update(const int& r, const int& c) {
    ++S[C[size] = c];
    D[size] = D[c]; U[D[c]] = size;
    U[size] = c; D[c] = size;
    if(H[r] < 0) H[r] = L[size] = R[size] = size;
    else {
      R[size] = R[H[r]]; L[R[H[r]]] = size;
      L[size] = H[r]; R[H[r]] = size;
    }
    X[size++] = r;
  }
  void init(int col) {
    for(int i = 0; i <= col; i++) {
      S[i] = 0; D[i] = U[i] = i;
      L[i+1] = i; R[i] = i + 1;
    }
    R[col] = 0;
    size = col + 1;
    memset(H, -1, sizeof(H));
    ret = (1 << 30);
  }
}dlx;
```

4.10 Spanning Tree On Directed Graph

```
int zhuliu(int root, int n, int m, Edge e[]) {
  int res = 0, u, v;
  while (true) {
    for (int i = 0; i < n; i++)
      in[i] = inf;
    for (int i = 0; i < m; i++)
      if (e[i].u != e[i].v && e[i].cost < in[e[i].v]) {
        pre[e[i].v] = e[i].u;
        in[e[i].v] = e[i].cost;
      }
    for (int i = 0; i < n; i++)
      if (i != root)
        if (in[i] == inf) return -1;
    int tn = 0;
    mst(id, -1); mst(visit, -1);
    in[root] = 0;
    for (int i = 0; i < n; i++) {
      res += in[i]; v = i;
      while (visit[v] != i && id[v] == -1 && v != root) {
        visit[v] = i; v = pre[v];
      }
      if(v != root && id[v] == -1) {
        for(int u = pre[v] ; u != v ; u = pre[u])
          id[u] = tn;
        id[v] = tn++;
      }
    }
    if(tn == 0) break;
    for (int i = 0; i < n; i++)
      if (id[i] == -1)
        id[i] = tn++;
    for (int i = 0; i < m;) {
      int v = e[i].v;
      e[i].u = id[e[i].u]; e[i].v = id[e[i].v];
      if (e[i].u != e[i].v)
        e[i++].cost -= in[v];
      else
        swap(e[i], e[--m]);
    }
    n = tn; root = id[root];
  }
  return res;
}
```

4.11 Blossom

```
void Push(int u) {
  Queue[Tail] = u; Tail++; InQueue[u] = true;
}
int Pop() {
  int res = Queue[Head]; Head++; return res;
}
int FindCommonAncestor(int u,int v) {
  memset(InPath, false, sizeof(InPath));
  while(true) {
    u = Base[u]; InPath[u] = true;
    if(u == Start) break;
    u = Father[Match[u]];
  }
  while(true) {
    v = Base[v];
    if(InPath[v])break;
    v = Father[Match[v]];
  }
  return v;
}
void ResetTrace(int u) {
  int v;
  while(Base[u] != NewBase) {
    v = Match[u];
    InBlossom[Base[u]] = InBlossom[Base[v]] = true;
    u = Father[v];
    if(Base[u] != NewBase) Father[u] = v;
  }
}
void BloosomContract(int u, int v) {
  NewBase = FindCommonAncestor(u, v);
  memset(InBlossom, false, sizeof(InBlossom));
  ResetTrace(u); ResetTrace(v);
  if(Base[u] != NewBase) Father[u] = v;
  if(Base[v] != NewBase) Father[v] = u;
  for(int tu = 1; tu <= N; tu++)
    if(InBlossom[Base[tu]]) {
      Base[tu] = NewBase;
      if(!InQueue[tu]) Push(tu);
```

```
    }
}
void FindAugmentingPath() {
  mst(InQueue, 0); mst(Father, 0);
  for(int i = 1; i <= N; i++)
    Base[i] = i;
  Head = Tail = 1; Push(Start); Finish = 0;
  while(Head < Tail) {
    int u = Pop();
    for(int v = 1; v <= N; v++)
      if(Graph[u][v] && (Base[u] != Base[v]) && (Match[u]
          != v)) {
        if((v == Start) || ((Match[v] > 0) &&
            Father[Match[v]] > 0))
          BloosomContract(u, v);
        else if(Father[v] == 0) {
          Father[v] = u;
          if(Match[v] > 0)
            Push(Match[v]);
          else {
            Finish = v;
            return;
          }
        }
      }
  }
}
void AugmentPath() {
  int u, v, w;
  u = Finish;
  while(u > 0) {
    v = Father[u]; w = Match[v];
    Match[v] = u; Match[u] = v;
    u = w;
  }
}
void Edmonds() {
  memset(Match, 0, sizeof(Match));
  for(int u = 1; u <= N; u++)
    if(Match[u] == 0) {
      Start = u;
      FindAugmentingPath();
      if(Finish > 0)AugmentPath();
    }
}
int getMatch() {
  Edmonds();
  Count = 0;
  for(int u = 1; u <= N; u++)
    if(Match[u] > 0)
      Count++;
  return Count / 2;
}
bool g[MAXN][MAXN];
int main() {
  int m;
  while(scanf("%d%d", &N, &m) == 2) {
    mst(g, 0); mst(Graph, 0);
    int u, v;
    for(int i = 1; i <= m; i++) {
      scanf("%d%d", &u, &v);
      p[i] = make_pair(u, v);
      g[u][v] = true; g[v][u] = true;
      Graph[u][v] = true; Graph[v][u] = true;
    }
    int cnt0 = getMatch();
  }
  return 0;
}
```

4.12 Clique

```
bool array[maxnum][maxnum];
bool use[maxnum];
int cn, bestn, p, e;
void dfs(int i) {
  int j;
  bool flag;
  if(i > p) {
    bestn = cn;
    printf("%d\n", bestn);
    for(j = 1; j <= p; j++)
      if(use[j])
        printf("%d ", j);
    printf("\n");
    return ;
  }
  flag = true;
  for(j = 1; j < i; j++)
    if(use[j] && !array[j][i]) {
      flag = false;
      break;
    }
  if(flag) {
    cn++; use[i] = true; dfs(i + 1); cn--;
  }
  if(cn + p - i > bestn) {
    use[i] = false; dfs(i + 1);
  }
}
scanf("%d%d", &u, &v);
array[u][v] = true; array[v][u] = true;
cn = bestn = 0; dfs(1);
```

4.13 Count The Number Of Spanning Tree

```
Matrix laplacian;
laplacian.clear();
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    if (i != j && G[i][j]) {
      laplacian.a[i][j] = -1;
      laplacian.a[i][i]++;
    }
printf("%d\n",laplacian.det(n..1));
```

5 Math

5.1 Cantor

```
1324 0 * 3! + 1 * 2! + 0 * 1! = 2
```

5.2 FastWalshHadamard Transform

```
void tf(int a[], int l, int r) {
  if(l + 1 == r) return ;
  int len = (r - l) >> 1, mid = l + len;
  tf(a, l, mid);
  tf(a, mid, r);
  for(int i = l; i < mid; i++) {
    a[i] += a[i + len];
    a[i] %= mod;
  }
}
void utf(int a[], int l, int r) {
  if(l + 1 == r)return ;
  int len = (r - l) >> 1, mid = l + len;
  for(int i = l; i < mid; i++) {
    a[i] = a[i] + mod - a[i + len];
    a[i] %= mod;
  }
  utf(a, l, mid);
  utf(a, mid, r);
}

AND
tf(A) = (tf(A0) + tf(A1), tf(A1))
utf(A) = (utf(A0) - utf(A1), utf(A1))

XOR
tf(A) = (tf(A0) + tf(A1), tf(A0) - tf(A1))
```

```
utf(A) = (utf((A0 + A1) / 2), utf((A0 - A1) / 2))

OR
tf(A) = (tf(A0), tf(A0) + tf(A1))
utf(A) = (utf(A0), utf(A1) - utf(A0))
```

5.3 FFT

```
complex<double> x1[N];
void change(complex<double> y[], int len) {
  for(int i = 1, j = len / 2; i < len - 1; i++) {
    if(i < j) swap(y[i], y[j]);
    int k = len / 2;
    for(; k <= j; k >>= 1) j -= k;
    if(j < k)j += k;
  }
}
void fft(complex<double> y[], int len, int on) {
  change(y, len);
  for(int h = 2; h <= len; h <<= 1) {
    complex<double> wn(cos(-on * 2 * PI / h), sin(-on * 2 *
        PI / h));
    for(int j = 0; j < len; j += h) {
      complex<double> w(1, 0);
      for(int k = j; k < j + h / 2; k++) {
        complex<double> u = y[k];
        complex<double> t = w * y[k + h / 2];
        y[k] = u + t;
        y[k + h / 2] = u - t;
        w = w * wn;
      }
    }
  }
}
//x1[i]=complex<double>(b[i], 0); fft(x1, n, 1);
//x1[i]=complex<double>(1,0)/x1[i];fft(x1,n,-1);
//x1[i]/=n;c[i]=x1[i].real();//Cyclic matrix
```

5.4 Fibonacci Sequence

```
long long f(long long x) {
  if(Fib.count(x)) return Fib[x];
  return Fib[x] = (f((x + 1) / 2) * f(x / 2) + f((x - 1) /
      2) * f((x - 2) / 2)) % MOD;
}
```

5.5 Integer Partition

```
void init() {
  f[0] = 1;
  int i, j, k, flag = 1;
  for(i = 1; i <= 100000; i++) {
    f[i] = 0;
    for(j = 1; ; j++) {
      int t = i - j * (3 * j - 1) / 2;
      int tt = i - j * (3 * j + 1) / 2;
      if(j & 1) flag = 1;
      else flag = -1;
      if(t < 0 && tt < 0) break;
      if(t >= 0) f[i] = (f[i] + flag * f[t]) % mod;
      if(tt >= 0) f[i] = (f[i] + flag * f[tt]) % mod;
    }
    f[i] = (f[i] + mod) % mod;
  }
}
```

5.6 Miller-Rabin

```
inline u32 fb(u32 n) {
    u32 k = 31,c = n - 1;
    while(!(c & (1U << k))) k--;
    return k;
}
bool witness(u32 a,u32 nbits,u32 n) {
    u32 d = 1,n1 = n-1;
    for(int i=nbits; i>=0; i--) {
        u32 x = d;
        d = mul(d,d,n);
        if (d == 1 && x != 1 && x != n1) return true;
        if (n1 & (1U << i)) d = mul(a,d,n);
    }
    return d != 1;
}
bool isprime(u32 n) {
    int nbits = fb(n);
    if (witness(2,nbits,n)) return false;
    if (n == 61) return true;
    if (witness(61,nbits,n)) return false;
    if (n < 916327) return true;
    if (witness(7,nbits,n)) return false;
    return true;
}
 //2, 325, 9375, 28178, 450775, 9780504, 1795265022
```

5.7 Mul Mod

```
LL mul_mod(LL x, LL y, LL n) {// x*y % n
  LL T = floor(sqrt(n) + 0.5);
  LL t = T * T - n;
  LL a = x / T, b = x % T;
  LL c = y / T, d = y % T;
  LL e = a * c / T, f = a * c % T;
  LL v = ((a * d + b * c) % n + e * t) % n;
  LL g = v / T, h = v % T;
  LL ret = (((f + g) * t % n + b * d) % n + h * T) % n;
  return (ret % n + n) % n;
}
long long mul(long long a,long long b) {
  return ((a * b-(long long)((long double)a * b / mod) *
      mod) % mod + mod) % mod;
}
```

5.8 NTT

```
const int P = 880803841;
const int G = 26;
const int NUM = 24;
struct solve {
  long long wn[NUM];
  void GetWn() {
    for(int i = 0; i < NUM; i++) {
      int t = 1 << i;
      wn[i] = quick_mod(G, (P - 1) / t, P);
    }
  }
  void Rader(long long a[], int len) {
    int j = len >> 1;
    for(int i = 1; i < len - 1; i++) {
      if(i < j) swap(a[i], a[j]);
      int k = len >> 1;
      while(j >= k) {
        j -= k;
        k >>= 1;
      }
      if(j < k) j += k;
    }
  }
  void NTT(long long a[], int len, int on) {
    Rader(a, len);
    int id = 0;
    for(int h = 2; h <= len; h <<= 1) {
      id++;
      for(int j = 0; j < len; j += h) {
        long long w = 1;
        for(int k = j; k < j + h / 2; k++) {
          long long u = a[k] % P;
          long long t = w * (a[k + h / 2] % P) % P;
          a[k] = (u + t) % P;
          a[k + h / 2] = ((u - t) % P + P) % P;
          w = w * wn[id] % P;
        }
```

```
    }
  }
  if(on == -1) {
    for(int i = 1; i < len / 2; i++)
      swap(a[i], a[len - i]);
    long long Inv = quick_mod(len, P - 2, P);
    for(int i = 0; i < len; i++)
      a[i] = a[i] % P * Inv % P;
  }
 }
 void Conv(long long a[], long long b[], int n) {
  NTT(a, n, 1);
  NTT(b, n, 1);
  for(int i = 0; i < n; i++)
    a[i] = a[i] * b[i] % P;
  NTT(a, n, -1);
 }
} ntt;
ntt.GetWn();
ntt.Conv(A, B, 1 << 18);//twice length
```

5.9 Pollard-Rho

```
LL Pollard_Rho(LL n, LL c = 1) {// get a factor of n in log(n)
 LL i = 1, k = 2, x = rand() % (n - 1) + 1, y = x, d;
 while(1) {
  i++;
  x = (mul_mod(x, x, n) + c) % n;
  d = __gcd(n, y - x);
  if(d > 1 && d < n) return d;
  if(y == x) return n;
  if(i == k) {
   k <<= 1;
   y = x;
  }
 }
}
```

5.10 Primitive Root

```
struct PR {
 int divs[N + 5];
 int primitive_root(const int p) {
  if (p == 2) return 1;
  int cnt = 0, m = p - 1;
  for (int i = 2; i * i <= m; ++ i) if (m%i == 0) {
      divs[cnt++] = i;
      if (i * i < m) divs[cnt++] = m / i;
    }
  int r = 2, j = 0;
  while (1) {
    for (j = 0; j < cnt; ++ j) {
      if (fastpow(r, divs[j], p) == 1) break;
    }
    if (j >= cnt) return r;
    r ++;
  }
  return -1;
 }
} pr_solver;
```

5.11 Romberg

```
double f(double x) {
 return x * x;
}
const int MAXREPT = 10;
const double eps = 1e-5;
double y[MAXREPT];
double Romberg(double aa, double bb) {
 int m, n;
 double h, x, s, q, ep, p;
 h = bb - aa;
 y[0] = h * (f(aa) + f(bb)) / 2.0;
 m = n = 1;
 ep = eps + 1.0;
 while ((ep > eps) && (m < MAXREPT)) {
  p = 0.0;
  for(int i = 0; i < n; i++)
    x = aa + (i + 0.5) * h, p += f(x);
  p = (y[0] + h * p) / 2.0;
  s = 1.0;
  for(int k = 1; k <= m; k++) {
    s *= 4.0;
    q = (s * p - y[k - 1]) / (s - 1.0);
    y[k - 1] = p;
    p = q;
  }
  p = fabs(q - y[m - 1]);
  y[m++] = q;
  n <<= 1;
  h /= 2.0;
 }
 return q;
}
```

6 nsqrt(n)

6.1 example

```
struct Re {
 int l, r, idx;
 Re() {}
 Re(int l, int r, int idx): l(l), r(r), idx(idx) {}
 bool operator < (const Re& re)const {
  if(l / 173 == re.l / 173) return r < re.r;
  return l / 173 < re.l / 173;
 }
};

for(int i = 0; i < n; i += limit) {
 ll[i / limit] = i;
 rr[i / limit] = min(i + limit - 1, n - 1);
 trie[i / limit].query();
}
l--;
r--;
int L = l / limit, R = r / limit;
for(int i = L; i <= R; i++) {
 if(l <= ll[i] && rr[i] <= r);
 else {
  int x = max(l, ll[i]), y = min(r, rr[i]);
  for(int j = x; j <= y; j++);
 }
}

int dfs(int u, int fa) {
 dfn[u] = ++dfs_idx;
 int sz = 0;
 for(int tmp = head[u]; ~tmp; tmp = edge[tmp].next) {
  int v = edge[tmp].to;
  if(v == fa) continue;
  update(v, u);
  C[v] = edge[tmp].w;
  sz += dfs(v, u);
  if(sz >= limit) {
   lab_idx++;
   for(int i = 0; i < sz; i++)
    lab[st[st_top--]] = lab_idx;
   sz = 0;
  }
 }
 st[++st_top] = u;
 return sz + 1;
}
```

7 String

7.1 KMP

```
f[0] = f[1] = 0;
for(i = 1; i < l1; i++) {
  j = f[i];
  while(j && P[i] != P[j]) j = f[j];
  f[i + 1] = (P[i] == P[j] ? j + 1: 0);
}
for(i = 0; i < l2; i++) {
  while(j && T[i] != P[j]) j = f[j];
  if(T[i] == P[j]) j++;
  if(j == l1) {
    ret++;
    j = f[j];
  }
}
```

7.2 ExKMP

```
void exkmp(char s[], char t[]) {
  int i, j, p, L;
  int lens = strlen(s); int lent = strlen(t);
  next[0] = lent;
  j = 0;
  while(j + 1 < lent && t[j] == t[j + 1]) j++;
  next[1] = j;
  int a = 1;
  for(i = 2; i < lent; i++) {
    p = next[a] + a - 1; L = next[i - a];
    if(i + L < p + 1)next[i] = L;
    else {
      j = max(0, p - i + 1);
      while(i + j < lent && t[i + j] == t[j]) j++;
      next[i] = j; a = i;
    }
  }
  j = 0;
  while(j < lens && j < lent && s[j] == t[j]) j++;
  extend[0] = j; a = 0;
  for(i = 1; i < lens; i++) {
    p = extend[a] + a - 1; L = next[i - a];
    if(L + i < p + 1) extend[i] = L;
    else {
      j = max(0, p - i + 1);
      while(i + j < lens && j < lent && s[i + j] == t[j])
          j++;
      extend[i] = j; a = i;
    }
  }
}//lcp(A[i..lenA-1], B)
```

7.3 Suffix Array

```
struct SuffixArray {
  int s[N], sa[N], rk[N], height[N], t[N], t2[N], c[N], n;
  void clear() {
    n = 0;
  }
  void build_sa(int m) {
    int *x = t, *y = t2;
    for(int i = 0; i < m; i++) c[i] = 0;
    for(int i = 0; i < n; i++) c[x[i] = s[i]]++;
    for(int i = 1; i < m; i++) c[i] += c[i - 1];
    for(int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
    for(int k = 1; k <= n; k <<= 1) {
      int p = 0;
      for(int i = n - k; i < n; i++) y[p++] = i;
      for(int i = 0; i < n; i++) if(sa[i] >= k)y[p++] =
          sa[i] - k;
      for(int i = 0; i < m; i++) c[i] = 0;
      for(int i = 0; i < n; i++) c[x[y[i]]]++;
      for(int i = 1; i < m; i++) c[i] += c[i - 1];
      for(int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] =
          y[i];
      swap(x, y);
      p = 1;
      x[sa[0]] = 0;
      for(int i = 1; i < n; i++)
        x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1]
            + k] == y[sa[i] + k]? p - 1: p++;
      if(p >= n) break;
      m = p;
    }
  }
  void build_height() {
    for(int i = 0; i < n; i++) rk[sa[i]] = i;
    for(int i = 0, k = 0; i < n - 1; i++) {
      if(k) k--;
      int j = sa[rk[i] - 1];
      while(s[i + k] == s[j + k]) k++;
      height[rk[i]] = k;
    }
  }
  void add(int ch) {
    s[n++] = ch;
  }
};
SuffixArray sa;
sa.clear();
int len = strlen(str);
for(int i = 0; i < len; i++) {
  sa.add(str[i] - 'a' + 1);
}
sa.add(0);
sa.build_sa(27);
int callo(int x, int len) {
  int l = 0,r = x;
  while(l + 1 < r) {
    int mid = (l + r) >> 1;
    if(callcp(mid + 1, x) >= len) r = mid;
    else l = mid;
  }
  return r;
}
int calup(int x, int len) {
  int l = x, r = sa.n;
  while(l + 1 < r) {
    int mid = (l + r) >> 1;
    if(callcp(x + 1, mid) >= len) l = mid;
    else r = mid;
  }
  return l;
}
for(int i = 1; i < sa.n; i++) {
  lcp[i][0] = sa.height[i];
}
for(int j = 1; j < 20; j++) {
  for(int i = 1; i + (1 << j) <= sa.n; i++) {
    lcp[i][j]=min(lcp[i][j - 1], lcp[i + (1 << (j - 1))][j
        - 1]);
  }
}
int callcp(int a, int b) {
  int tt = b - a + 1, len = Log[tt];
  return min(lcp[a][len], lcp[a + tt - (1 << len)][len]);
}
```

7.4 Aho-Corasick Automaton

```
void insert(char str[]) {
  int len = strlen(str);
  int now = 0;
  for(int i = 0; i < len; i++) {
    int ind = str[i] - 'a';
    if(!trie[now][ind]) {
      total++;
      memset(trie[total], 0, sizeof(trie[total]));
      trie[now][ind] = total;
    }
    now = trie[now][ind];
  }
  flag[now]++;
```

```
}
void AC() {
  front = rear = 0;
  for(int i = 0; i < 26; i++) {
    if(trie[0][i]) {
      que[rear++] = trie[0][i];
      f[trie[0][i]] = 0;
    }
  }
  while(front < rear) {
    int u = que[front++];
    for(i = 0; i < 26; i++) {
      if(trie[u][i]) {
        int v = trie[u][i];
        que[rear++] = v;
        f[v] = trie[f[u]][i];
      } else trie[u][i] = trie[f[u]][i];
    }
  }
}
void AC() {
  queue<int> que;
  for(map<int,int>::iterator it = trie[0].begin(); it !=
       trie[0].end(); it++) {
    que.push(it->second);
    f[it->second] = 0;
  }
  while(!que.empty()) {
    int u = que.front();
    que.pop();
    for(map<int,int>::iterator it = trie[u].begin(); it !=
         trie[u].end(); it++) {
      int v = it->second;
      que.push(v);
      int x = f[u];
      for(; x && !trie[x].count(it->first); x = f[x]);
      f[v] = trie[x][it->first];
    }
  }
}
```

7.5 Minimum Representation

```
int MinimumRepresentation(int *s, int l) {
  int l=strlen(s);
  int i = 0, j = 1, k = 0;
  while (1) {
    if (i + k >= l || j + k >= l) break;
    if (s[i + k] == s[j + k]) {
      k++;
      continue;
    } else {
      if (s[j + k] > s[i + k]) j += k + 1;
      else i += k + 1;
      k = 0;
      if (i == j) j++;
    }
  }
  return min(i, j);
}
```

7.6 Manacher

```
int add(int l, int len) {
  if(len <= 0) return 0;
  long long x = (h[l] - h[l + len] * p[len] % mod + mod) %
       mod;
  if(vis[x]) return vis[x];
  else {
    int now = ++idx;
    int fa = add(l + 1, len - 2);
    g[fa].push_back(now);
    return vis[x] = now;
  }
}
void Manacher(int len) {
  tot = 0;
  a[tot++] = -1; a[tot++] = 0;
  int len = strlen(s);
  for(int i = 0; i < len; i++) {
    a[tot++] = s[i] - 'a' + 1; a[tot++] = 0;
  }
  a[tot] = 0;
  for(int i = 1, mx = 0, id; i < tot; i++) {
    if(mx > i) cc[i] = min(cc[2 * id - i], mx - i);
    else cc[i] = 1;
    for(; a[i + cc[i]] == a[i - cc[i]]; cc[i]++);
    if(cc[i] + i > mx) mx = cc[i] + i, id = i;
  }
}
for(int i = 1; i < tot; i++) {
  int len = cc[i] - 1, l = (i - 1) / 2 - len / 2;
  cnt[add(l, len)]++;
}
```

7.7 Palindromic Tree

```
struct Palindromic_Tree {
  int next[MAXN][N], fail[MAXN], cnt[MAXN], num[MAXN],
       len[MAXN], S[MAXN];
  int last, n, p;
  int newnode(int l) {
    for(int i = 0; i < N; ++i) next[p][i] = 0;
    cnt[p] = 0; num[p] = 0; len[p] = l;
    return p++;
  }
  void init() {
    p = 0; newnode(0) ; newnode(-1);
    last = 0; n = 0; S[n] = -1; fail[0] = 1;
  }
  int get_fail(int x) {
    while(S[n - len[x] - 1] != S[n]) x = fail[x];
    return x;
  }
  void add(int c) {
    c -= 'a';
    S[++n] = c;
    int cur = get_fail(last);
    if(!next[cur][c]) {
      int now = newnode(len[cur] + 2);
      fail[now] = next[get_fail(fail[cur])][c];
      next[cur][c] = now;
      num[now] = num[fail[now]] + 1;
    }
    last = next[cur][c];
    cnt[last]++;
  }
  void count() {
    for(int i = p - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
  }
} T;

void init ( int n ) {
  p = 0 ; newnode ( 0 ) ; newnode ( -1 ) ;
  clr ( S , -1 ) ; L = n ; R = n - 1 ;
  last[0] = last[1] = 1 ; fail[0] = 1 ; tot = 0 ;
}
int get_fail ( int v , bool d ) {
  if ( d ) while ( S[R - len[v] - 1] != S[R] ) v = fail[v] ;
  else while ( S[L + len[v] + 1] != S[L] ) v = fail[v] ;
  return v ;
}
void add ( int c , bool d ) {
  if ( d ) S[++ R] = c ;
  else S[-- L] = c ;
  int cur = get_fail ( last[d] , d ) ;
  if ( !nxt[cur][c] ) {
    int now = newnode ( len[cur] + 2 ) ;
    fail[now] = nxt[get_fail ( fail[cur] , d )][c] ;
    nxt[cur][c] = now ;
    num[now] = num[fail[now]] + 1 ;
  }
  last[d] = nxt[cur][c] ;
  if ( len[last[d]] == R - L + 1 ) last[d ^ 1] = last[d] ;
  tot += num[last[d]] ;
}
```

7.8 SAM

```
if(p->go[x]) {
  cc++; p = p->go[x];
} else {
  while(p && !p->go[x])p = p->pre;
  if(0 == p) {
    p = root; cc = 0;
  } else {
    cc = p->step + 1; p = p->go[x];
  }
}
struct State {
  State *pre, *go[26], *dp[20], *next[26];
  int step, mxstep, val;
  long long les;
  void clear() {
    pre = 0;
    step = val = 0;
    mst(go, 0); mst(dp, 0); mst(next, 0);
  }
}*root, *last;
State statePool[N * 2], *b[2 * N], *cur;
int c[N];
void init() {
  cur = statePool; root = last = cur++;
  root->clear();
}
void extend(int w) {
  State *p = last; State *np = cur++;
  np->clear();
  np->step = np->mxstep = p->step + 1;
  np->val = 1;
  while(p && !p->go[w])
    p->go[w] = np, p = p->pre;
  if(p == 0)
    np->pre = root;
  else {
    State *q = p->go[w];
    if(p->step + 1 == q->step)
      np->pre = q;
    else {
      State *nq = cur++;
      nq->clear();
      memcpy(nq->go, q->go, sizeof(q->go));
      nq->step = p->step + 1;
      nq->pre = q->pre;
      nq->mxstep = q->mxstep;
      q->pre = nq;
      np->pre = nq;
      while(p && p->go[w] == q)
        p->go[w] = nq, p = p->pre;
    }
  }
  last = np;
}
void dfs(State* p, long long& ret) {
  if(p->pre) ret += 1LL * p->val * (p->step - p->pre->step);
  p->les = ret;
  for(int i = 0; i < 26; i++) {
    if(p->next[i])dfs(p->next[i], ret);
  }
}
int main() {
  scanf("%s", s);
  int len = strlen(s);
  reverse(s, s + len);
  init();
  for(int i = 0; i < len; i++) {
    extend(s[i] - 'A'); accept[i] = last;
  }
  for(State *p = statePool; p < cur; p++) {
    p->dp[0] = p->pre; c[p->step]++;
  }
  for(int i = 1; i <= len; i++) c[i] += c[i - 1];
  for(State *p = cur-1; p >= statePool; p--)
       b[--c[p->step]] = p;
  for(int i = 0; i < cur - statePool; i++) {
    State *p = b[i];
    for(int i = 1; i < 20; i++) {
      p->dp[i] = (0 == p->dp[i - 1]? 0: p->dp[i - 1]->dp[i
           - 1]);
    }
  }
  for(int i = cur - statePool - 1; i; i--) {
    State *p = b[i];
    p->pre->val += p->val;
    p->pre->next[s[p->mxstep - p->pre->step - 1] - 'A'] = p;
  }
  ret = 0;
  dfs(root, ret);
  int q;
  scanf("%d", &q);
  for(int i = 0; i < q; i++) {
    int x, y;
    scanf("%d%d",&x, &y);
    x = len - x;
    y = len - y;
    swap(x, y);
    State *now = accept[y];
    int len = (y - x + 1);
    for(int i = 19; i >= 0; i--) {
      if(now->dp[i] && now->dp[i]->step >= len) {
        now = now->dp[i];
      }
    }
    long long ans = now->les - 1LL * (now->step - len + 1)
         * now->val;
    printf("%I64d\n", ans);
  }
  return 0;
}
```

8 Tree

8.1 Divide And Conquer Tree

```
int sz[N], vis[N], fa[N], que[N];
int TT;
int n, q;
int getroot(int u, int& tot) {
  int mi = n, root = 0;
  int l = 0, r = 0;
  que[r++] = u;
  fa[u] = 0;
  while(l < r) {
    u = que[l++];
    for(int i = 0; i < g[u].size(); i++) {
      int v = g[u][i];
      if(v == fa[u] || vis[v] == TT)continue;
      que[r++] = v;
      fa[v] = u;
    }
  }
  tot = r;
  for(--l; l >= 0; l--) {
    int x = que[l], ma = 0;
    sz[x] = 1;
    for(int i = 0; i < g[x].size(); i++) {
      int v = g[x][i];
      if(v == fa[x] || vis[v] == TT)continue;
      ma = max(ma, sz[v]);
      sz[x] += sz[v];
    }
    ma = max(ma, r - sz[x]);
    if(ma < mi) {
      mi = ma, root = x;
    }
  }
  return root;
}
int dis[N];
void go(int u, int rr) {
  int l = 0, r = 0;
  que[r++] = u;
  fa[u] = 0;
  dis[u] = 1;
```

```
  while(l < r) {
    u = que[l++];
    re[u].push_back(Re(rr, idx_of_bit, dis[u]));
    for(int i = 0; i < g[u].size(); i++) {
      int v = g[u][i];
      if(v == fa[u] || vis[v] == TT)continue;
      que[r++] = v;
      fa[v] = u;
      dis[v] = dis[u] + 1;
    }
  }
  bit[idx_of_bit].tree.resize(r + 1, 0);
  for(int i = 0; i < r; i++) {
    int u = que[i];
    bit[idx_of_bit].update(dis[u], w[u]);
    bit[rr].update(dis[u], w[u]);
  }
}
void solve(int u) {
  int tot;
  int root = getroot(u, tot);
  vis[root] = TT;
  int r = ++idx_of_bit;
  re[root].push_back(Re(r, 0, 0));
  bit[r].tree.resize(tot + 1, 0);
  bit[r].update(0, w[root]);
  int tmp = root;
  for(int i = 0; i < g[tmp].size(); i++) {
    int v = g[tmp][i];
    if(vis[v] == TT)continue;
    ++idx_of_bit;
    go(v, r);
  }
  for(int i = 0; i < g[tmp].size(); i++) {
    int v = g[tmp][i];
    if(vis[v] == TT)continue;
    solve(v);
  }
}

//dfs
void predfs(int u, int fa) {
  sz[u] = 1;
  for(int v: g[u]) {
    if(v==fa||vis[v])continue;
    predfs(v,u);
    sz[u]+=sz[v];
  }
}
int root;
int mx[N], mi;
void dfsroot(int u, int fa, int r) {
  mx[u] = sz[r] - sz[u];
  for(int v: g[u]) {
    if(v == fa || vis[v]) continue;
    dfsroot(v, u, r);
    mx[u] = max(mx[u], sz[v]);
  }
  if(mx[u] < mi) {
    mi = mx[u]; root = u;
  }
}
void cal(int u, int fa, Re pre) {
  long long s = 0,ss = 0;
  for(int i = 0; i < K; i++) {
    s += (6 - pre.s[i] - cnt[root][i]) % 3 * p3[i];
    ss += pre.s[i] * p3[i];
  }
  ret += mp[s];
  re.push_back(ss);
  for(int v: g[u]) {
    if(v == fa || vis[v]) continue;
    Re now;
    for(int j = 0; j < K; j++)
      now.s[j] = (pre.s[j] + cnt[v][j]) % 3;
    cal(v, u, now);
  }
}
void dfs(int u) {
  mp.clear(); mp[0] = 1;
  for(int v: g[u]) {
    if(vis[v])continue;
    re.clear();
    Re now;
    for(int j = 0; j < K; j++)
      now.s[j] = cnt[v][j];
    cal(v, u, now);
    for(int j = 0; j < re.size(); j++)
      mp[re[j]]++;
  }
}
void gao(int u) {
  mi = n; root = u;
  predfs(u, 0); dfsroot(u, 0, u);
  vis[root]=true; dfs(root);
  int tmp = root;
  for(int v: g[tmp]) {
    if(vis[v])continue;
    gao(v);
  }
}
```

8.2 Heavy-Light Decomposition

```
void init() {
  memset(head, -1, sizeof(head));
  countedge = 0;
  loc = 0;
  mst(tp, -1); mst(fa, -1); mst(son, -1); mst(w, -1);
}
void dfs(int u) {
  sz[u] = 1;
  for(int tmp = head[u]; tmp != -1; tmp = edge[tmp].next) {
    int v = edge[tmp].to;
    if(v == fa[u])continue;
    fa[v] = u; dep[v] = dep[u] + 1;
    dfs(v);
    if(-1 == son[u] || sz[v] > sz[son[u]])son[u]=v;
    sz[u] += sz[v];
  }
}
void build_tree(int u, int f) {
  w[u] = ++loc; tp[u] = f;
  if(son[u] != -1) {
    build_tree(son[u], f);
  }
  for(int tmp = head[u]; tmp != -1; tmp = edge[tmp].next) {
    int v = edge[tmp].to;
    if(v == son[u] || v == fa[u]) {
      continue;
    }
    build_tree(v, v);
  }
}
int solve(int x, int y) {
  int f1 = tp[x], f2 = tp[y], ret = 0;
  while(f1 != f2) {
    if(dep[f1] < dep[f2]) {
      swap(f1, f2); swap(x, y);
    }
    ret = max(ret, query(1, 1, loc, w[f1], w[x]));
    x = fa[f1]; f1 = tp[x];
  }
  if(x == y)return ret;
  if(dep[x] > dep[y])swap(x, y);
  return max(ret, query(1, 1, loc, w[son[x]], w[y]));
}
dep[1] = 0;
dfs(1);
build_tree(1, 1);
for(int i = 1; i < n; i++) {
  if(dep[re[i][0]] > dep[re[i][1]])swap(re[i][0], re[i][1]);
  update(1, 1, loc, w[re[i][1]], re[i][2]);
}
void solve(int x, int y, int val) {
  int f1 = tp[x], f2 = tp[y], ret = 0;
  while(f1 != f2) {
    if(dep[f1] < dep[f2]) {
      swap(f1, f2); swap(x, y);
    }
```

```
    update(1, 1, loc, w[f1], w[x], val);
    x = fa[f1]; f1 = tp[x];
  }
  if(dep[x] > dep[y])swap(x, y);
  update(1, 1, loc, w[x], w[y], val);
}

void prepare() {/*****bfs*****/
  dep[0] = 0; fa[1] = 0;
  f = r = 0;
  que[r++] = 1;
  for(; f<r; ) {
    int u = que[f++]; dep[u] = dep[fa[u]] + 1;
    for(int tmp = head[u]; ~tmp; tmp = edge[tmp].next) {
      int v = edge[tmp].to;
      if(v != fa[u]) {
        fa[v] = u; que[r++] = v;
      }
    }
  }
  for(int i = r-1; i >= 0; i--) {
    int u = que[i]; sz[u]++;
    if(-1 == son[fa[u]] || sz[u] > sz[son[fa[u]]]) {
      son[fa[u]] = u;
    }
    sz[fa[u]] += sz[u];
  }
}
void build_tree() {
  deque<pair<int, int> > que;
  que.push_front( {1, 1});
  while(!que.empty()) {
    pair<int, int> p = que.front();
    que.pop_front();
    int u = p.first, f = p.second;
    w[u] = ++loc; tp[u] = f;
    if(son[u] != -1) {
      que.push_front( {son[u], f});
    }
    for(int tmp = head[u]; ~tmp; tmp = edge[tmp].next) {
      int v = edge[tmp].to;
      if(v != fa[u] && v != son[u]) {
        que.push_back( {v, v});
      }
    }
  }
}
```

9 Others

9.1 bash Script

```
while true; do
       ./gen > input
       ./sol < input > output.sol
       ./bf < input > output.bf

       diff output.sol output.bf
       if [ $? -ne 0 ] ; then break; fi
done
```

9.2 tool

```
int __size__ = 256 << 20;
char * __p__ = (char *) malloc(__size__) + __size__;
__asm__("movl %0,%%esp\n"::"r"(__p__));

template <class T>
inline bool scan_d(T &ret) {
  char c;
  int sgn;
  if(c = getchar(), c == EOF) return 0; //EOF
  while(c != '-' && (c < '0' || c > '9')) c = getchar();
  sgn = (c == '-')? -1: 1;
  ret = (c == '-')? 0: (c - '0');
  while(c = getchar(), c >= '0' &&c <= '9') ret = ret * 10
       + (c - '0');
  ret *= sgn;
  return 1;
}

for(int y = 0; y < n; y++) D(x) += d(x, y);
(n - 1) * d(x, y) + D(y) - D(x)

(2k-1)^2 n*(4*n*n-1)/3
(2*k-1)^3 n*n*(2*n*n-1)
k*(k+1) n*(n+1)*(n+2)/3
k*(k+1)*(k+2) n*(n+1)*(n+2)*(n+3)/4
(1+.....+n)1
(1+...+n^2)1/2 1/2
(1+...+n^3)1/6 1/2 1/3

set cindent autoindent number sts=4 sw=4 ts=4 et
set background=dark
set backspace=indent,eol,start
set autochdir
syntax on
map <F4> :!g++ -g -std=c++11 % -o %<<cr>
map <F5> :!%<<cr>
```

9.3 java

```
Scanner cin = new Scanner(new File("derangements.in"));
PrintWriter cout = new PrintWriter(new
     File("derangements.out"));
```

9.4 Minimum Representation

```
string str1, str2;
string min_pre(string str) {
  vector<string> box;
  string ret = "";
  int equal = 0, st = 0;
  for(int i = 0; i < str.size(); i++) {
    if(str[i] == '0') equal++;
    else equal--;
    if(equal == 0) {
      if(i - 1 > st + 1) {
        box.push_back("0" + min_pre(str.substr(st + 1,i - 1
             - st)) + "1");
      } else box.push_back("01");
      st = i + 1;
    }
  }
  sort(box.begin(), box.end());
  for(int i = 0; i < box.size(); i++) ret += box[i];
  return ret;
}
while(cin >> str1 >> str2) if(min_pre(str1) ==
     min_pre(str2)) printf("same\n");
```