Contents

1 计算几何

1.1 二维计算几何基本操作

```cpp
const double PI = 3.14159265358979323846264338327950288;
double arcSin(const double &a) {
 return (a <= -1.0) ? (-PI / 2) : ((a >= 1.0) ? (PI / 2) : (asin(a))); }
double arcCos(const double &a) {
 return (a <= -1.0) ? (PI) : ((a >= 1.0) ? (0) : (acos(a))); }
struct point { double x, y; // something omitted
  point rot(const double &a) const { // counter-clockwise
   return point(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a)); }
  point rot90() const { return point(-y, x); } // counter-clockwise
  point project(const point &p1, const point &p2) const {
   const point &q = *this; return p1 + (p2 - p1) * (dot(p2 - p1, q - p1) / (p2 - p1).norm()); }
  bool onSeg(const point &a, const point &b) const { // a, b inclusive
   const point &c = *this; return sign(dot(a - c, b - c)) <= 0 && sign(det(b - a, c - a)) == 0; }
  double distLP(const point &p1, const point &p2) const { // dist from *this to line p1->p2
   const point &q = *this; return fabs(det(p2 - p1, q - p1)) / (p2 - p1).len(); }
  double distSP(const point &p1, const point &p2) const { // dist from *this to segment [p1, p2]
    const point &q = *this;
    if (dot(p2 - p1, q - p1) < EPS) return (q - p1).len();
    if (dot(p1 - p2, q - p2) < EPS) return (q - p2).len();
    return distLP(p1, p2);
  }
  bool inAngle(const point &p1, const point &p2) const { // det(p1, p2) ≥ 0
    const point &q = *this; return det(p1, q) > -EPS && det(p2, q) < EPS;
  }
};
bool lineIntersect(const point &a, const point &b, const point &c, const point &d, point &e) {
  double s1 = det(c - a, d - a), s2 = det(d - b, c - b);
  if (!sign(s1 + s2)) return false; e = (b - a) * (s1 / (s1 + s2)) + a; return true;
}
int segIntersectCheck(const point &a, const point &b, const point &c, const point &d, point &o) {
  static double s1, s2, s3, s4;
  static int iCnt;
  int d1 = sign(s1 = det(b - a, c - a)), d2 = sign(s2 = det(b - a, d - a));
  int d3 = sign(s3 = det(d - c, a - c)), d4 = sign(s4 = det(d - c, b - c));
  if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {
    o = (c * s2 - d * s1) / (s2 - s1); return true;
  } iCnt = 0;
  if (d1 == 0 && c.onSeg(a, b)) o = c, ++iCnt;
  if (d2 == 0 && d.onSeg(a, b)) o = d, ++iCnt;
  if (d3 == 0 && a.onSeg(c, d)) o = a, ++iCnt;
  if (d4 == 0 && b.onSeg(c, d)) o = b, ++iCnt;
  return iCnt ? 2 : 0; // 不相交返回 0, 严格相交返回 1, 非严格相交返回 2
}
struct circle {
  point o; double r, rSqure;
  bool inside(const point &a) { return (a - o).len() < r + EPS; } // 非严格
  bool contain(const circle &b) const { return sign(b.r + (o - b.o).len() - r) <= 0; } // 非严格
  bool disjunct(const circle &b) const { return sign(b.r + r - (o - b.o).len()) <= 0; } // 非严格
  int isCL(const point &p1, const point &p2, point &a, point &b) const {
    double x = dot(p1 - o, p2 - p1), y = (p2 - p1).norm();
    double d = x * x - y * ((p1 - o).norm() - rSqure);
    if (d < -EPS) return 0; if (d < 0) d = 0;
    point q1 = p1 - (p2 - p1) * (x / y);
    point q2 = (p2 - p1) * (sqrt(d) / y);
    a = q1 - q2; b = q1 + q2; return q2.len() < EPS ? 1 : 2;
  }
  int tanCP(const point &p, point &a, point &b) const { // 返回切点, 注意可能与 p 重合
    double x = (p - o).norm(), d = x - rSqure;
    if (d < -EPS) return 0; if (d < 0) d = 0;
    point q1 = (p - o) * (rSqure / x), q2 = ((p - o) * (-r * sqrt(d) / x)).rot90();
    a = o + (q1 - q2); b = o + (q1 + q2); return q2.len() < EPS ? 1 : 2;
  }
};
bool checkCrossCS(const circle &cir, const point &p1, const point &p2) { // 非严格
  const point &c = cir.o; const double &r = cir.r;
  return c.distSP(p1, p2) < r + EPS && (r < (c - p1).len() + EPS || r < (c - p2).len() + EPS);
}
bool checkCrossCC(const circle &cir1, const circle &cir2) { // 非严格
  double &r1 = cir1.r, &r2 = cir2.r, d = (cir1.o - cir2.o).len();
  return d < r1 + r2 + EPS && fabs(r1 - r2) < d + EPS;
}
int isCC(const circle &cir1, const circle &cir2, point &a, point &b) {
  const point &c1 = cir1.o, &c2 = cir2.o;
  double x = (c1 - c2).norm(), y = ((cir1.rSqure - cir2.rSqure) / x + 1) / 2;
  double d = cir1.rSqure / x - y * y;
  if (d < -EPS) return 0; if (d < 0) d = 0;
  point q1 = c1 + (c2 - c1) * y, q2 = ((c2 - c1) * sqrt(d)).rot90();
  a = q1 - q2; b = q1 + q2; return q2.len() < EPS ? 1 : 2;
}
vector<pair<point, point> > tanCC(const circle &cir1, const circle &cir2) {
// 注意: 如果只有三条切线, 即 s1 = 1, s2 = 1, 返回的切线可能重复, 切点没有问题
  vector<pair<point, point> > list;
  if (cir1.contain(cir2) || cir2.contain(cir1)) return list;
  const point &c1 = cir1.o, &c2 = cir2.o;
  double r1 = cir1.r, r2 = cir2.r; point p, a1, b1, a2, b2; int s1, s2;
  if (sign(r1 - r2) == 0) {
    p = c2 - c1; p = (p * (r1 / p.len())).rot90();
    list.push_back(make_pair(c1 + p, c2 + p)); list.push_back(make_pair(c1 - p, c2 - p));
  } else {
    p = (c2 * r1 - c1 * r2) / (r1 - r2);
    s1 = cir1.tanCP(p, a1, b1); s2 = cir2.tanCP(p, a2, b2);
    if (s1 >= 1 && s2 >= 1)
      list.push_back(make_pair(a1, a2)), list.push_back(make_pair(b1, b2));
  } p = (c1 * r2 + c2 * r1) / (r1 + r2);
  s1 = cir1.tanCP(p, a1, b1); s2 = cir2.tanCP(p, a2, b2);
  if (s1 >= 1 && s2 >= 1)
    list.push_back(make_pair(a1, a2)), list.push_back(make_pair(b1, b2));
  return list;
}
bool distConvexPIn(const point &p1, const point &p2, const point &p3, const point &p4, const point &q) {
```

```
  point o12 = (p1 - p2).rot90(), o23 = (p2 - p3).rot90(), o34 = (p3 - p4).rot90();
  return (q - p1).inAngle(o12, o23) || (q - p3).inAngle(o23, o34)
    || ((q - p2).inAngle(o23, p3 - p2) && (q - p3).inAngle(p2 - p3, o23));
}
double distConvexP(int n, point ps[], const point &q) { // 外部点到多边形的距离
  int left = 0, right = n; while (right - left > 1) { int mid = (left + right) / 2;
    if (distConvexPIn(ps[(left + n - 1) % n], ps[left], ps[mid], ps[(mid + 1) % n], q))
      right = mid; else left = mid;
  } return q.distSP(ps[left], ps[right % n]);
}
double areaCT(const circle &cir, point pa, point pb) {
  pa = pa - cir.o; pb = pb - cir.o; double R = cir.r;
  if (pa.len() < pb.len()) swap(pa, pb); if (pb.len() < EPS) return 0;
  point pc = pb - pa; double a = pa.len(), b = pb.len(), c = pc.len(), S, h, theta;
  double cosB = dot(pb, pc) / b / c, B = acos(cosB);
  double cosC = dot(pa, pb) / a / b, C = acos(cosC);
  if (b > R) {
    S = C * 0.5 * R * R; h = b * a * sin(C) / c;
    if (h < R && B < PI * 0.5) S -= acos(h / R) * R * R - h * sqrt(R * R - h * h);
  } else if (a > R) {
    theta = PI - B - asin(sin(B) / R * b);
    S = 0.5 * b * R * sin(theta) + (C - theta) * 0.5 * R * R;
  } else S = 0.5 * sin(C) * b * a;
  return S;
}
circle minCircle(const point &a, const point &b) {
  return circle((a + b) * 0.5, (b - a).len() * 0.5);
}
circle minCircle(const point &a, const point &b, const point &c) { // 钝角三角形没有被考虑
  double a2( (b - c).norm() ), b2( (a - c).norm() ), c2( (a - b).norm() );
  if (b2 + c2 <= a2 + EPS) return minCircle(b, c);
  if (a2 + c2 <= b2 + EPS) return minCircle(a, c);
  if (a2 + b2 <= c2 + EPS) return minCircle(a, b);
  double A = 2.0 * (a.x - b.x), B = 2.0 * (a.y - b.y);
  double D = 2.0 * (a.x - c.x), E = 2.0 * (a.y - c.y);
  double C = a.norm() - b.norm(), F = a.norm() - c.norm();
  point p((C * E - B * F) / (A * E - B * D), (A * F - C * D) / (A * E - B * D));
  return circle(p, (p - a).len());
}
circle minCircle(point P[], int N) { // 1-based
  if (N == 1) return circle(P[1], 0.0);
  random_shuffle(P + 1, P + N + 1); circle O = minCircle(P[1], P[2]);
  Rep(i, 1, N) if(!O.inside(P[i])) { O = minCircle(P[1], P[i]);
    Foru(j, 1, i) if(!O.inside(P[j])) { O = minCircle(P[i], P[j]);
      Foru(k, 1, j) if(!O.inside(P[k])) O = minCircle(P[i], P[j], P[k]); }
  } return O;
}
```

1.2 圆的面积模板

```
struct Event { point p; double alpha; int add; // 构造函数省略
  bool operator < (const Event &other) const { return alpha < other.alpha; } };
void circleKCover(circle *c, int N, double *area) { // area[k]: 至少被覆盖 k 次
  static bool overlap[MAXN][MAXN], g[MAXN][MAXN];
  Rep(i, 0, N + 1) area[i] = 0.0; Rep(i, 1, N) Rep(j, 1, N) overlap[i][j] = c[i].contain(c[j]);
  Rep(i, 1, N) Rep(j, 1, N) g[i][j] = !(overlap[i][j] || overlap[j][i] || c[i].disjunct(c[j]));
  Rep(i, 1, N) { static Event events[MAXN * 2 + 1]; int totE = 0, cnt = 1;
    Rep(j, 1, N) if (j != i && overlap[j][i]) ++cnt;
    Rep(j, 1, N) if (j != i && g[i][j]) {
      circle &a = c[i], &b = c[j]; double l = (a.o - b.o).norm();
      double s = ((a.r - b.r) * (a.r + b.r) / l + 1) * 0.5;
      double t = sqrt(-(l - sqr(a.r - b.r)) * (l - sqr(a.r + b.r)) / (l * l * 4.0));
      point dir = b.o - a.o, nDir = point(-dir.y, dir.x);
      point aa = a.o + dir * s + nDir * t;
      point bb = a.o + dir * s - nDir * t;
      double A = atan2(aa.y - a.o.y, aa.x - a.o.x);
      double B = atan2(bb.y - a.o.y, bb.x - a.o.x);
      events[totE++] = Event(bb, B, 1); events[totE++] = Event(aa, A, -1); if (B > A) ++cnt;
    } if (totE == 0) { area[cnt] += PI * c[i].rSquare; continue; }
    sort(events, events + totE); events[totE] = events[0];
    Foru(j, 0, totE) {
      cnt += events[j].add; area[cnt] += 0.5 * det(events[j].p, events[j + 1].p);
      double theta = events[j + 1].alpha - events[j].alpha; if (theta < 0) theta += 2.0 * PI;
      area[cnt] += 0.5 * c[i].rSquare * (theta - sin(theta));
}}}
```

1.3 多边形相关

```
struct Polygon { // stored in [0, n)
  int n; point list[MAXN];
  Polygon cut(const point &a, const point &b) {
    static Polygon res;
    static point o;
    res.n = 0;
    for (int i = 0; i < n; ++i) {
      int s1 = sign(det(list[i] - a, b - a));
      int s2 = sign(det(list[(i + 1) % n] - a, b - a));
      if (s1 <= 0) res.list[res.n++] = list[i];
      if (s1 * s2 < 0) {
        lineIntersect(a, b, list[i], list[(i + 1) % n], o);
        res.list[res.n++] = o;
      }
    } return res;
  }
  bool contain(const point &p) const { // 1 if on border or inner, 0 if outter
    static point A, B;
    int res = 0;
    for (int i = 0; i < n; ++i) {
      A = list[i]; B = list[(i + 1) % n];
      if (p.onSeg(A, B)) return 1;
      if (sign(A.y - B.y) <= 0) swap(A, B);
      if (sign(p.y - A.y) > 0) continue;
```

```
      if (sign(p.y - B.y) <= 0) continue;
      res += (int)(sign(det(B - p, A - p)) > 0);
    }
    return res & 1;
  }
  bool convexContain(const point &p) const { // sort by polar angle
    for (int i = 1; i < n; ++i) list[i] = list[i] - list[0];
    point q = p - list[0];
    if (sign(det(list[1], q)) < 0 || sign(det(list[n - 1], q)) > 0) return false;
    int l = 2, r = n - 1;
    while (l <= r) {
      int mid = (l + r) >> 1;
      double d1 = sign(det(list[mid], q)), d2 = sign(det(list[mid - 1], q));
      if (d1 <= 0) {
        if (d2 <= 0) {
          if (sign(det(q - list[mid - 1], list[mid] - list[mid - 1]) <= 0) <= 0)
            return true;
        } else r = mid - 1;
      } else l = mid + 1;
    }
    return false;
  }
  double isPLAtan2(const point &a, const point &b) {
    double k = (b - a).alpha(); if (k < 0) k += 2 * PI;
    return k;
  }
  point isPL_Get(const point &a, const point &b, const point &s1, const point &s2) {
    double k1 = det(b - a, s1 - a), k2 = det(b - a, s2 - a);
    if (sign(k1) == 0) return s1;
    if (sign(k2) == 0) return s2;
    return (s1 * k2 - s2 * k1) / (k2 - k1);
  }
  int isPL_Dic(const point &a, const point &b, int l, int r) {
    int s = (det(b - a, list[l] - a) < 0) ? -1 : 1;
    while (l <= r) {
      int mid = (l + r) / 2;
      if (det(b - a, list[mid] - a) * s <= 0) r = mid - 1;
      else l = mid + 1;
    }
    return r + 1;
  }
  int isPL_Find(double k, double w[]) {
    if (k <= w[0] || k > w[n - 1]) return 0;
    int l = 0, r = n - 1, mid;
    while (l <= r) {
      mid = (l + r) / 2;
      if (w[mid] >= k) r = mid - 1;
      else l = mid + 1;
    } return r + 1;
  }
  bool isPL(const point &a, const point &b, point &cp1, point &cp2) {  // O(logN)
    static double w[MAXN * 2]; // pay attention to the array size
    for (int i = 0; i <= n; ++i) list[i + n] = list[i];
    for (int i = 0; i < n; ++i) w[i] = w[i + n] = isPLAtan2(list[i], list[i + 1]);
    int i = isPL_Find(isPLAtan2(a, b), w);
    int j = isPL_Find(isPLAtan2(b, a), w);
    double k1 = det(b - a, list[i] - a), k2 = det(b - a, list[j] - a);
    if (sign(k1) * sign(k2) > 0) return false; // no intersection
    if (sign(k1) == 0 || sign(k2) == 0) { // intersect with a point or a line in the convex
      if (sign(k1) == 0) {
        if (sign(det(b - a, list[i + 1] - a)) == 0) cp1 = list[i], cp2 = list[i + 1];
        else cp1 = cp2 = list[i];
        return true;
      }
      if (sign(k2) == 0) {
        if (sign(det(b - a, list[j + 1] - a)) == 0) cp1 = list[j], cp2 = list[j + 1];
        else cp1 = cp2 = list[j];
      }
      return true;
    }
    if (i > j) swap(i, j);
    int x = isPL_Dic(a, b, i, j), y = isPL_Dic(a, b, j, i + n);
    cp1 = isPL_Get(a, b, list[x - 1], list[x]);
    cp2 = isPL_Get(a, b, list[y - 1], list[y]);
    return true;
  }
  double getI(const point &O) const {
    if (n <= 2) return 0;
    point G(0.0, 0.0);
    double S = 0.0, I = 0.0;
    for (int i = 0; i < n; ++i) {
      const point &x = list[i], &y = list[(i + 1) % n];
      double d = det(x, y);
      G = G + (x + y) * d / 3.0;
      S += d;
    } G = G / S;
    for (int i = 0; i < n; ++i) {
      point x = list[i] - G, y = list[(i + 1) % n] - G;
      I += fabs(det(x, y)) * (x.norm() + dot(x, y) + y.norm());
    }
    return I = I / 12.0 + fabs(S * 0.5) * (O - G).norm();
  }
};
```

1.4 半平面交

```
struct Border {
  point p1, p2; double alpha;
  Border() : p1(), p2(), alpha(0.0) {}
  Border(const point &a, const point &b): p1(a), p2(b), alpha( atan2(p2.y - p1.y, p2.x - p1.x) ) {}
  bool operator == (const Border &b) const { return sign(alpha - b.alpha) == 0; }
  bool operator < (const Border &b) const {
    int c = sign(alpha - b.alpha); if (c != 0) return c > 0;
    return sign(det(b.p2 - b.p1, p1 - b.p1)) >= 0;
  }
};
point isBorder(const Border &a, const Border &b) { // a and b should not be parallel
  point is; lineIntersect(a.p1, a.p2, b.p1, b.p2, is); return is;
}
bool checkBorder(const Border &a, const Border &b, const Border &me) {
  point is; lineIntersect(a.p1, a.p2, b.p1, b.p2, is);
```

```
    return sign(det(me.p2 - me.p1, is - me.p1)) > 0;
}
double HPI(int N, Border border[]) {
    static Border que[MAXN * 2 + 1]; static point ps[MAXN];
    int head = 0, tail = 0, cnt = 0; // [head, tail)
    sort(border, border + N); N = unique(border, border + N) - border;
    for (int i = 0; i < N; ++i) {
        Border &cur = border[i];
        while (head + 1 < tail && !checkBorder(que[tail - 2], que[tail - 1], cur)) --tail;
        while (head + 1 < tail && !checkBorder(que[head], que[head + 1], cur)) ++head;
        que[tail++] = cur;
    } while (head + 1 < tail && !checkBorder(que[tail - 2], que[tail - 1], que[head])) --tail;
    while (head + 1 < tail && !checkBorder(que[head], que[head + 1], que[tail - 1])) ++head;
    if (tail - head <= 2) return 0.0;
    Foru(i, head, tail) ps[cnt++] = isBorder(que[i], que[(i + 1 == tail) ? (head) : (i + 1)]);
    double area = 0; Foru(i, 0, cnt) area += det(ps[i], ps[(i + 1) % cnt]);
    return fabs(area * 0.5); // or (-area * 0.5)
}
```

1.5 最大面积空凸包

```
inline bool toUpRight(const point &a, const point &b) {
    int c = sign(b.y - a.y); if (c > 0) return true;
    return c == 0 && sign(b.x - a.x) > 0;
}
inline bool cmpByPolarAngle(const point &a, const point &b) { // counter-clockwise, shorter first if they share the same polar angle
    int c = sign(det(a, b)); if (c != 0) return c > 0;
    return sign(b.len() - a.len()) > 0;
}
double maxEmptyConvexHull(int N, point p[]) {
    static double dp[MAXN][MAXN];
    static point vec[MAXN];
    static int seq[MAXN]; // empty triangles formed with (0,0), vec[o], vec[seq[i]]
    double ans = 0.0;
    Rep(o, 1, N) {
        int totVec = 0;
        Rep(i, 1, N) if (toUpRight(p[o], p[i])) vec[++totVec] = p[i] - p[o];
        sort(vec + 1, vec + totVec + 1, cmpByPolarAngle);
        Rep(i, 1, totVec) Rep(j, 1, totVec) dp[i][j] = 0.0;
        Rep(k, 2, totVec) {
            int i = k - 1;
            while (i > 0 && sign( det(vec[k], vec[i]) ) == 0) --i;
            int totSeq = 0;
            for (int j; i > 0; i = j) {
                seq[++totSeq] = i;
                for (j = i - 1; j > 0 && sign(det(vec[i] - vec[k], vec[j] - vec[k])) > 0; --j);
                double v = det(vec[i], vec[k]) * 0.5;
                if (j > 0) v += dp[i][j];
                dp[k][i] = v;
                cMax(ans, v);
            } for (int i = totSeq - 1; i >= 1; --i) cMax( dp[k][ seq[i] ], dp[k][seq[i + 1]] );
        }
    } return ans;
}
```

1.6 最近点对

```
int N; point p[maxn];
bool cmpByX(const point &a, const point &b) { return sign(a.x - b.x) < 0; }
bool cmpByY(const int &a, const int &b) { return p[a].y < p[b].y; }
double minimalDistance(point *c, int n, int *ys) {
    double ret = 1e+20;
    if (n < 20) {
        Foru(i, 0, n) Foru(j, i + 1, n) cMin(ret, (c[i] - c[j]).len() );
        sort(ys, ys + n, cmpByY); return ret;
    } static int mergeTo[maxn];
    int mid = n / 2; double xmid = c[mid].x;
    ret = min(minimalDistance(c, mid, ys), minimalDistance(c + mid, n - mid, ys + mid));
    merge(ys, ys + mid, ys + mid, ys + n, mergeTo, cmpByY);
    copy(mergeTo, mergeTo + n, ys);
    Foru(i, 0, n) {
        while (i < n && sign(fabs(p[ys[i]].x - xmid) - ret) > 0) ++i;
        int cnt = 0;
        Foru(j, i + 1, n)
            if (sign(p[ys[j]].y - p[ys[i]].y - ret) > 0) break;
            else if (sign(fabs(p[ys[j]].x - xmid) - ret) <= 0) {
                ret = min(ret, (p[ys[i]] - p[ys[j]]).len());
                if (++cnt >= 10) break;
            }
    } return ret;
}
double work() {
    sort(p, p + n, cmpByX); Foru(i, 0, n) ys[i] = i; return minimalDistance(p, n, ys);
}
```

1.7 凸包与点集直径

```
vector<point> convexHull(int n, point ps[]) { // counter-clockwise, strict
    static point qs[MAXN * 2];
    sort(ps, ps + n, cmpByXY);
    if (n <= 2) return vector(ps, ps + n);
    int k = 0;
    for (int i = 0; i < n; qs[k++] = ps[i++])
        while (k > 1 && det(qs[k - 1] - qs[k - 2], ps[i] - qs[k - 1]) < EPS) --k;
    for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
        while (k > t && det(qs[k - 1] - qs[k - 2], ps[i] - qs[k - 1]) < EPS) --k;
    return vector<point>(qs, qs + k);
}
double convexDiameter(int n, point ps[]) {
    if (n < 2) return 0; if (n == 2) return (ps[1] - ps[0]).len();
    double k, ans = 0;
    for (int x = 0, y = 1, nx, ny; x < n; ++x) {
        for(nx = (x == n - 1) ? (0) : (x + 1); ; y = ny) {
            ny = (y == n - 1) ? (0) : (y + 1);
            if ( sign(k = det(ps[nx] - ps[x], ps[ny] - ps[y])) <= 0) break;
        } ans = max(ans, (ps[x] - ps[y]).len());
```

```
    if (sign(k) == 0) ans = max(ans, (ps[x] - ps[ny]).len());
  } return ans;
}
```

1.8 Farmland

```
struct node { int begin[MAXN], *end; } a[MAXN]; // 按对 p[i] 的极角的 atan2 值排序
bool check(int n, point p[], int b1, int b2, bool vis[MAXN][MAXN]) {
  static pii l[MAXN * 2 + 1]; static bool used[MAXN];
  int tp(0), *k, p, p1, p2; double area(0.0);
  for (l[0] = pii(b1, b2); ; ) {
    vis[p1 = l[tp].first][p2 = l[tp].second] = true;
    area += det(p[p1], p[p2]);
    for (k = a[p2].begin; k != a[p2].end; ++k) if (*k == p1) break;
    k = (k == a[p2].begin) ? (a[p2].end - 1) : (k - 1);
    if ((l[++tp] = pii(p2, *k)) == l[0]) break;
  } if (sign(area) < 0 || tp < 3) return false;
  Rep(i, 1, n) used[i] = false;
  for (int i = 0; i < tp; ++i) if (used[p = l[i].first]) return false; else
      used[p] = true;
  return true; // a face with tp vertices
}
int countFaces(int n, point p[]) {
  static bool vis[MAXN][MAXN]; int ans = 0;
  Rep(x, 1, n) Rep(y, 1, n) vis[x][y] = false;
  Rep(x, 1, n) for (int *itr = a[x].begin; itr != a[x].end; ++itr) if (!vis[x
      ][*itr])
    if (check(n, p, x, *itr, vis)) ++ans;
  return ans;
}
```

1.9 Voronoi 图

不能有重点, 点数应当不小于 2

```
#define Oi(e) ((e)->oi)
#define Dt(e) ((e)->dt)
#define On(e) ((e)->on)
#define Op(e) ((e)->op)
#define Dn(e) ((e)->dn)
#define Dp(e) ((e)->dp)
#define Other(e, p) ((e)->oi == p ? (e)->dt : (e)->oi)
#define Next(e, p) ((e)->oi == p ? (e)->on : (e)->dn)
#define Prev(e, p) ((e)->oi == p ? (e)->op : (e)->dp)
#define V(p1, p2, u, v) (u = p2->x - p1->x, v = p2->y - p1->y)
#define C2(u1, v1, u2, v2) (u1 * v2 - v1 * u2)
#define C3(p1, p2, p3) ((p2->x - p1->x) * (p3->y - p1->y) - (p2->y - p1->y) *
      (p3->x - p1->x))
#define Dot(u1, v1, u2, v2) (u1 * u2 + v1 * v2)
#define dis(a,b) (sqrt( (a->x - b->x) * (a->x - b->x) + (a->y - b->y) * (a->y
      - b->y) ))
const int maxn = 110024;
const int aix = 4;
const double eps = 1e-7;
int n, M, k;
struct gEdge {
  int u, v; double w;
  bool operator <(const gEdge &e1) const { return w < e1.w - eps; }
} E[aix * maxn], MST[maxn];
struct point {
  double x, y; int index; edge *in;
  bool operator <(const point &p1) const { return x < p1.x - eps || (abs(x -
      p1.x) <= eps && y < p1.y - eps); }
};
struct edge { point *oi, *dt; edge *on, *op, *dn, *dp; };

point p[maxn], *Q[maxn];
edge mem[aix * maxn], *elist[aix * maxn];
int nfree;
void Alloc_memory() { nfree = aix * n; edge *e = mem; for (int i = 0; i <
    nfree; i++) elist[i] = e++; }
void Splice(edge *a, edge *b, point *v) {
  edge *next;
  if (Oi(a) == v) next = On(a), On(a) = b; else next = Dn(a), Dn(a) = b;
  if (Oi(next) == v) Op(next) = b; else Dp(next) = b;
  if (Oi(b) == v) On(b) = next, Op(b) = a; else Dn(b) = next, Dp(b) = a;
}
edge *Make_edge(point *u, point *v) {
  edge *e = elist[--nfree];
  e->on = e->op = e->dn = e->dp = e; e->oi = u; e->dt = v;
  if (!u->in) u->in = e;
  if (!v->in) v->in = e;
  return e;
}
edge *Join(edge *a, point *u, edge *b, point *v, int side) {
  edge *e = Make_edge(u, v);
  if (side == 1) {
    if (Oi(a) == u) Splice(Op(a), e, u);
    else Splice(Dp(a), e, u);
    Splice(b, e, v);
  } else {
    Splice(a, e, u);
    if (Oi(b) == v) Splice(Op(b), e, v);
    else Splice(Dp(b), e, v);
  } return e;
}
void Remove(edge *e) {
  point *u = Oi(e), *v = Dt(e);
  if (u->in == e) u->in = e->on;
  if (v->in == e) v->in = e->dn;
  if (Oi(e->on) == u) e->on->op = e->op; else e->on->dp = e->op;
  if (Oi(e->op) == u) e->op->on = e->on; else e->op->dn = e->on;
  if (Oi(e->dn) == v) e->dn->op = e->dp; else e->dn->dp = e->dp;
  if (Oi(e->dp) == v) e->dp->on = e->dn; else e->dp->dn = e->dn;
  elist[nfree++] = e;
}
void Low_tangent(edge *e_l, point *o_l, edge *e_r, point *o_r, edge **l_low,
    point **OL, edge **r_low, point **OR) {
  for (point *d_l = Other(e_l, o_l), *d_r = Other(e_r, o_r); ; )
    if (C3(o_l, o_r, d_l) < -eps)      e_l = Prev(e_l, d_l), o_l = d_l, d_l =
        Other(e_l, o_l);
    else if (C3(o_l, o_r, d_r) < -eps) e_r = Next(e_r, d_r), o_r = d_r, d_r =
        Other(e_r, o_r);
    else break;
  *OL = o_l, *OR = o_r; *l_low = e_l, *r_low = e_r;
}
void Merge(edge *lr, point *s, edge *rl, point *u, edge **tangent) {
  double l1, l2, l3, l4, r1, r2, r3, r4, cot_L, cot_R, u1, v1, u2, v2, n1,
      cot_n, P1, cot_P;
  point *O, *D, *OR, *OL; edge *B, *L, *R;
```

```
  Low_tangent(lr, s, rl, u, &L, &OL, &R, &OR);
  for (*tangent = B = Join(L, OL, R, OR, 0), O = OL, D = OR; ; ) {
    edge *El = Next(B, O), *Er = Prev(B, D), *next, *prev;
    point *l = Other(El, O), *r = Other(Er, D);
    V(l, O, l1, l2); V(l, D, l3, l4); V(r, O, r1, r2); V(r, D, r3, r4);
    double cl = C2(l1, l2, l3, l4), cr = C2(r1, r2, r3, r4);
    bool BL = cl > eps, BR = cr > eps;
    if (!BL && !BR) break;
    if (BL) {
      double dl = Dot(l1, l2, l3, l4);
      for (cot_L = dl / cl; ; Remove(El), El = next, cot_L = cot_n) {
        next = Next(El, O); V(Other(next, O), O, u1, v1); V(Other(next, O), D
            , u2, v2);
        n1 = C2(u1, v1, u2, v2); if (!(n1 > eps)) break;
        cot_n = Dot(u1, v1, u2, v2) / n1;
        if (cot_n > cot_L) break;
      }
    } if (BR) {
      double dr = Dot(r1, r2, r3, r4);
      for (cot_R = dr / cr; ; Remove(Er), Er = prev, cot_R = cot_P) {
        prev = Prev(Er, D); V(Other(prev, D), O, u1, v1); V(Other(prev, D), D
            , u2, v2);
        P1 = C2(u1, v1, u2, v2); if (!(P1 > eps)) break;
        cot_P = Dot(u1, v1, u2, v2) / P1;
        if (cot_P > cot_R) break;
      }
    } l = Other(El, O); r = Other(Er, D);
    if (!BL || (BL && BR && cot_R < cot_L)) B = Join(B, O, Er, r, 0), D = r;
    else B = Join(El, l, B, D, 0), O = l;
  }
}
void Divide(int s, int t, edge **L, edge **R) {
  edge *a, *b, *c, *ll, *lr, *rl, *rr, *tangent;
  int n = t - s + 1;
  if (n == 2) *L = *R = Make_edge(Q[s], Q[t]);
  else if (n == 3) {
    a = Make_edge(Q[s], Q[s + 1]), b = Make_edge(Q[s + 1], Q[t]);
    Splice(a, b, Q[s + 1]);
    double v = C3(Q[s], Q[s + 1], Q[t]);
    if (v > eps)       c = Join(a, Q[s], b, Q[t], 0), *L = a, *R = b;
    else if (v < -eps) c = Join(a, Q[s], b, Q[t], 1), *L = c, *R = c;
    else *L = a, *R = b;
  } else if (n > 3) {
    int split = (s + t) / 2;
    Divide(s, split, &ll, &lr); Divide(split + 1, t, &rl, &rr);
    Merge(lr, Q[split], rl, Q[split + 1], &tangent);
    if (Oi(tangent) == Q[s]) ll = tangent;
    if (Dt(tangent) == Q[t]) rr = tangent;
    *L = ll; *R = rr;
  }
}
void Make_Graph() {
  edge *start, *e; point *u, *v;
  for (int i = 0; i < n; i++) {
    start = e = (u = &p[i])->in;
    do{ v = Other(e, u);
      if (u < v) E[M++].u = (u - p, v - p, dis(u, v)); // M < aix * maxn
    } while ((e = Next(e, u)) != start);
  }
}
int b[maxn];
int Find(int x) { while (x != b[x]) { b[x] = b[b[x]]; x = b[x]; } return x; }
void Kruskal() {
  memset(b, 0, sizeof(b)); sort(E, E + M);
  for (int i = 0; i < n; i++) b[i] = i;
  for (int i = 0, kk = 0; i < M && kk < n - 1; i++) {
    int m1 = Find(E[i].u), m2 = Find(E[i].v);
    if (m1 != m2) b[m1] = m2, MST[kk++] = E[i];
  }
}
void solve() {
  scanf("%d", &n);
  for (int i = 0; i < n; i++) scanf("%lf%lf", &p[i].x, &p[i].y), p[i].index =
      i, p[i].in = NULL;
  Alloc_memory(); sort(p, p + n);
  for (int i = 0; i < n; i++) Q[i] = p + i;
  edge *L, *R; Divide(0, n - 1, &L, &R);
  M = 0; Make_Graph(); Kruskal();
}
int main() { solve(); return 0; }
```

1.10 三维计算几何基本操作

```
struct point { double x, y, z; // something omitted
  friend point det(const point &a, const point &b) {
    return point(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.
        y * b.x);
  }
  friend double mix(const point &a, const point &b, const point &c) {
    return a.x * b.y * c.z + a.y * b.z * c.x + a.z * b.x * c.y - a.z * b.y *
        c.x - a.x * b.z * c.y - a.y * b.x * c.z;
  }
  double distLP(const point &p1, const point &p2) const {
    return det(p2 - p1, *this - p1).len() / (p2 - p1).len();
  }
  double distFP(const point &p1, const point &p2, const point &p3) const {
    point n = det(p2 - p1, p3 - p1); return fabs( dot(n, *this - p1) / n.len
        () );
  }
};
double distLL(const point &p1, const point &p2, const point &q1, const point
    &q2) {
  point p = q1 - p1, u = p2 - p1, v = q2 - q1;
  double d = u.norm() * v.norm() - dot(u, v) * dot(u, v);
  if (sign(d) == 0) return p1.distLP(q1, q2);
  double s = (dot(p, u) * v.norm() - dot(p, v) * dot(u, v)) / d;
  return (p1 + u * s).distLP(q1, q2);
}
double distSS(const point &p1, const point &p2, const point &q1, const point
    &q2) {
  point p = q1 - p1, u = p2 - p1, v = q2 - q1;
  double d = u.norm() * v.norm() - dot(u, v) * dot(u, v);
  if (sign(d) == 0) return min( min((p1 - q1).len(), (p1 - q2).len()),
                    min((p2 - q1).len(), (p2 - q2).len()));
  double s1 = (dot(p, u) * v.norm() - dot(p, v) * dot(u, v)) / d;
  double s2 = (dot(p, v) * u.norm() - dot(p, u) * dot(u, v)) / d;
  if (s1 < 0.0) s1 = 0.0; if (s1 > 1.0) s1 = 1.0;
  if (s2 < 0.0) s2 = 0.0; if (s2 > 1.0) s2 = 1.0;
  point r1 = p1 + u * s1; point r2 = q1 + v * s2;
  return (r1 - r2).len();
```

```
}
bool isFL(const point &p, const point &o, const point &q1, const point &q2, point &res) {
  double a = dot(o, q2 - p), b = dot(o, q1 - p), d = a - b;
  if (sign(d) == 0) return false;
  res = (q1 * a - q2 * b) / d;
  return true;
}
bool isFF(const point &p1, const point &o1, const point &p2, const point &o2, point &a, point &b) {
  point e = det(o1, o2), v = det(o1, e);
  double d = dot(o2, v); if (sign(d) == 0) return false;
  point q = p1 + v * (dot(o2, p2 - p1) / d);
  a = q; b = q + e;
  return true;
}
```

1.11 凸多面体切割

```
vector<vector<point> > convexCut(const vector<vector<point> > &pss, const point &p, const point &o) {
  vector<vector<point> > res;
  vector<point> sec;
  for (unsigned itr = 0, size = pss.size(); itr < size; ++itr) {
    const vector<point> &ps = pss[itr];
    int n = ps.size();
    vector<point> qs;
    bool dif = false;
    for (int i = 0; i < n; ++i) {
      int d1 = sign( dot(o, ps[i] - p) );
      int d2 = sign( dot(o, ps[(i + 1) % n] - p) );
      if (d1 <= 0) qs.push_back(ps[i]);
      if (d1 * d2 < 0) {
        point q;
        isFL(p, o, ps[i], ps[(i + 1) % n], q); // must return true
        qs.push_back(q);
        sec.push_back(q);
      }
      if (d1 == 0) sec.push_back(ps[i]);
      else dif = true;
      dif |= dot(o, det(ps[(i + 1) % n] - ps[i], ps[(i + 2) % n] - ps[i])) < -EPS;
    }
    if (!qs.empty() && dif)
      res.insert(res.end(), qs.begin(), qs.end());
  }
  if (!sec.empty()) {
    vector<point> tmp( convexHull2D(sec, o) );
    res.insert(res.end(), tmp.begin(), tmp.end());
  }
  return res;
}

vector<vector<point> > initConvex() {
  vector<vector<point> > pss(6, vector<point>(4));
  pss[0][0] = pss[1][0] = pss[2][0] = point(-INF, -INF, -INF);
  pss[0][3] = pss[1][1] = pss[5][2] = point(-INF, -INF,  INF);
  pss[0][1] = pss[2][3] = pss[4][2] = point(-INF,  INF, -INF);
  pss[0][2] = pss[5][3] = pss[4][1] = point(-INF,  INF,  INF);
  pss[1][3] = pss[2][1] = pss[3][2] = point( INF, -INF, -INF);
  pss[1][2] = pss[5][1] = pss[3][3] = point( INF, -INF,  INF);
  pss[2][2] = pss[4][3] = pss[3][1] = point( INF,  INF, -INF);
  pss[5][0] = pss[4][0] = pss[3][0] = point( INF,  INF,  INF);
  return pss;
}
```

1.12 三维凸包

不能有重点

```
namespace ConvexHull3D {
  #define volume(a, b, c, d) (mix(ps[b] - ps[a], ps[c] - ps[a], ps[d] - ps[a]))
  vector<Facet> getHull(int n, point ps[]) {
    static int mark[MAXN][MAXN], a, b, c; int stamp = 0; bool exist = false;
    vector<Facet> facet; random_shuffle(ps, ps + n);
    for (int i = 2; i < n && !exist; i++) {
      point ndir = det(ps[0] - ps[i], ps[1] - ps[i]);
      if (ndir.len() < EPS) continue;
      swap(ps[i], ps[2]); for (int j = i + 1; j < n && !exist; j++)
        if (sign(volume(0, 1, 2, j)) != 0) {
          exist = true; swap(ps[j], ps[3]);
          facet.push_back(Facet(0, 1, 2)); facet.push_back(Facet(0, 2, 1));
        }
    } if (!exist) return ConvexHull2D(n, ps);
    for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j) mark[i][j] = 0;
    stamp = 0; for (int v = 3; v < n; ++v) {
      vector<Facet> tmp; ++stamp;
      for (unsigned i = 0; i < facet.size(); i++) {
        a = facet[i].a; b = facet[i].b; c = facet[i].c;
        if (sign(volume(v, a, b, c)) < 0)
          mark[a][b] = mark[a][c] = mark[b][a] = mark[b][c] = mark[c][a] = mark[c][b] = stamp;
        else tmp.push_back(facet[i]);
      } facet = tmp;
      for (unsigned i = 0; i < tmp.size(); i++) {
        a = facet[i].a; b = facet[i].b; c = facet[i].c;
        if (mark[a][b] == stamp) facet.push_back(Facet(b, a, v));
        if (mark[b][c] == stamp) facet.push_back(Facet(c, b, v));
        if (mark[c][a] == stamp) facet.push_back(Facet(a, c, v));
      }
    } return facet;
  }
  #undef volume
}
namespace Gravity {
  using ConvexHull3D::Facet;
  point findG(point ps[], const vector<Facet> &facet) {
    double ws = 0; point res(0.0, 0.0, 0.0), o = ps[ facet[0].a ];
    for (int i = 0, size = facet.size(); i < size; ++i) {
      const point &a = ps[ facet[i].a ], &b = ps[ facet[i].b ], &c = ps[ facet[i].c ];
      point p = (a + b + c + o) * 0.25; double w = mix(a - o, b - o, c - o);
      ws += w; res = res + p * w;
    } res = res / ws;
    return res;
  }
}
```

1.13 长方体表面点距离

```
int r;
void turn(int i, int j, int x, int y, int z, int x0, int y0, int L, int W,
    int H) {
  if (z == 0) r = min(r, x * x + y * y);
  else {
    if (i >= 0 && i < 2)  turn(i + 1, j, x0 + L + z, y, x0 + L - x, x0 + L,
        y0, H, W, L);
    if (j >= 0 && j < 2)  turn(i, j + 1, x, y0 + W + z, y0 + W - y, x0, y0 +
        W, L, H, W);
    if (i <= 0 && i > -2) turn(i - 1, j, x0 - z, y, x - x0, x0 - H, y0, H, W,
        L);
    if (j <= 0 && j > -2) turn(i, j - 1, x, y0 - z, y - y0, x0, y0 - H, L, H,
        W);
  }
}
int calc(int L, int H, int W, int x1, int y1, int z1, int x2, int y2, int z2)
    {
  if (z1 != 0 && z1 != H)
    if (y1 == 0 || y1 == W) swap(y1, z1), swap(y2, z2), swap(W, H);
    else                    swap(x1, z1), swap(x2, z2), swap(L, H);
  if (z1 == H) z1 = 0, z2 = H - z2;
  r = INF; turn(0, 0, x2 - x1, y2 - y1, z2, -x1, -y1, L, W, H);
  return r;
}
```

1.14 最小覆盖球

```
namespace MinBall {
int outCnt;
point out[4], res;
double radius;
void ball() {
  static point q[3];
  static double m[3][3], sol[3], L[3], det;
  int i, j;
  res = point(0.0, 0.0, 0.0);
  radius = 0.0;
  switch (outCnt) {
  case 1:
    res = out[0];
    break;
  case 2:
    res = (out[0] + out[1]) * 0.5;
    radius = (res - out[0]).norm();
    break;
  case 3:
    q[0] = out[1] - out[0];
    q[1] = out[2] - out[0];
    for (i = 0; i < 2; ++i)
      for (j = 0; j < 2; ++j)
        m[i][j] = dot(q[i], q[j]) * 2.0;
    for (i = 0; i < 2; ++i)
      sol[i] = dot(q[i], q[i]);
    det = m[0][0] * m[1][1] - m[0][1] * m[1][0];
    if (sign(det) == 0)
      return;
    L[0] = (sol[0] * m[1][1] - sol[1] * m[0][1]) / det;
    L[1] = (sol[1] * m[0][0] - sol[0] * m[1][0]) / det;
    res = out[0] + q[0] * L[0] + q[1] * L[1];
    radius = (res - out[0]).norm();
    break;
  case 4:
    q[0] = out[1] - out[0];
    q[1] = out[2] - out[0];
    q[2] = out[3] - out[0];
    for (i = 0; i < 3; ++i)
      for (j = 0; j < 3; ++j)
        m[i][j] = dot(q[i], q[j]) * 2;
    for (i = 0; i < 3; ++i)
      sol[i] = dot(q[i], q[i]);
    det = m[0][0] * m[1][1] * m[2][2] + m[0][1] * m[1][2] * m[2][0]
        + m[0][2] * m[2][1] * m[1][0] - m[0][2] * m[1][1] * m[2][0]
        - m[0][1] * m[1][0] * m[2][2] - m[0][0] * m[1][2] * m[2][1];
    if (sign(det) == 0)
      return;
    for (j = 0; j < 3; ++j) {
      for (i = 0; i < 3; ++i)
        m[i][j] = sol[i];
      L[j] = (m[0][0] * m[1][1] * m[2][2] + m[0][1] * m[1][2] * m[2][0]
          + m[0][2] * m[2][1] * m[1][0] - m[0][2] * m[1][1] * m[2][0]
          - m[0][1] * m[1][0] * m[2][2] - m[0][0] * m[1][2] * m[2][1])
          / det;
      for (i = 0; i < 3; ++i)
        m[i][j] = dot(q[i], q[j]) * 2;
    }
    res = out[0];
    for (i = 0; i < 3; ++i)
      res += q[i] * L[i];
    radius = (res - out[0]).norm();
  }
}

void minball(int n, point pt[]) {
  ball();
  if (outCnt < 4)
    for (int i = 0; i < n; ++i)
      if ((res - pt[i]).norm() > +radius + EPS) {
        out[outCnt] = pt[i];
        ++outCnt;
        minball(i, pt);
        --outCnt;
        if (i > 0) {
          point Tt = pt[i];
          memmove(&pt[1], &pt[0], sizeof(point) * i);
          pt[0] = Tt;
        }
      }
}

pair<point, double> main(int npoint, point pt[]) { // 0-based
  random_shuffle(pt, pt + npoint);
  radius = -1;
  for (int i = 0; i < npoint; i++) {
    if ((res - pt[i]).norm() > EPS + radius) {
      outCnt = 1;
      out[0] = pt[i];
      minball(i, pt);
    }
  }
```

```
  return make_pair(res, sqrt(radius));
}
}
```

1.15 三维向量操作矩阵

- 绕单位向量 $u = (u_x, u_y, u_z)$ 右手方向旋转 θ 度的矩阵:
$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta) - u_z\sin\theta & u_xu_z(1-\cos\theta) + u_y\sin\theta \\ u_yu_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta) - u_x\sin\theta \\ u_zu_x(1-\cos\theta) - u_y\sin\theta & u_zu_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$
$$= \cos\theta I + \sin\theta \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} + (1-\cos\theta) \begin{bmatrix} u_x^2 & u_xu_y & u_xu_z \\ u_yu_x & u_y^2 & u_yu_z \\ u_zu_x & u_zu_y & u_z^2 \end{bmatrix}$$
- 点 a 绕单位向量 $u = (u_x, u_y, u_z)$ 右手方向旋转 θ 度的对应点为 $a' = a\cos\theta + (u\times a)\sin\theta + (u\otimes u)a(1-\cos\theta)$
- 关于向量 v 作对称变换的矩阵 $H = I - 2\frac{vv^T}{v^Tv}$,
- 点 a 对称点: $a' = a - 2\frac{v^Ta}{v^Tv}\cdot v$

1.16 立体角

对于任意一个四面体 $OABC$, 从 O 点观察 ΔABC 的立体角 $\tan\frac{\Omega}{2} = \frac{\mathrm{mix}(\vec{a},\vec{b},\vec{c})}{|a||b||c| + (\vec{a}\cdot\vec{b})|c| + (\vec{a}\cdot\vec{c})|b| + (\vec{b}\cdot\vec{c})|a|}$.

2 数据结构

2.1 动态凸包 (只支持插入)

```
#define x first   // upperHull ← (x, y)
#define y second  // lowerHull ← (x, −y)
typedef map<int, int> mii;
typedef map<int, int>::iterator mit;
struct point { point(const mit &p): x(p->first), y(p->second) {} };
inline bool checkInside(mii &a, const point &p) { // border inclusive
  int x = p.x, y = p.y; mit p1 = a.lower_bound(x);
  if (p1 == a.end()) return false; if (p1->x == x) return y <= p1->y;
  if (p1 == a.begin()) return false; mit p2(p1--);
  return sign(det(p - point(p1), point(p2) - p)) >= 0;
} inline void addPoint(mii &a, const point &p) { // no collinear points
  int x = p.x, y = p.y; mit pnt = a.insert(make_pair(x, y)).first, p1, p2;
  for (pnt->y = y; ; a.erase(p2)) {
    p1 = pnt; if (++p1 == a.end()) break;
    p2 = p1;  if (++p1 == a.end()) break;
    if (det(point(p2) - p, point(p1) - p) < 0) break;
  } for ( ; ; a.erase(p2)) {
    if ((p1 = pnt) == a.begin()) break;
    if (--p1 == a.begin()) break; p2 = p1--;
    if (det(point(p2) - p, point(p1) - p) > 0) break;
  }
}
```

2.2 Rope 用法

```
#include <ext/rope>
using __gnu_cxx::crope; using __gnu_cxx::rope;
a = b.substr(from, len);          // [from, from + len)
a = b.substr(from);               // [from, from]
b.c_str();                        // might lead to memory leaks
b.delete_c_str();                 // delete the c_str that created before
a.insert(p, str);                 // insert str before position p
a.erase(i, n);                    // erase [i, i + n)
```

2.3 可持久化 Treap

```
inline bool randomBySize(int a, int b) {
  static long long seed = 1;
  return (seed = seed * 48271 % 2147483647) * (a + b) < 2147483647LL * a;
}
tree merge(tree x, tree y) {
  if (x == null) return y; if (y == null) return x;
  tree t = NULL;
  if (randomBySize(x->size, y->size)) t = newNode(x), t->r = merge(x->r, y);
  else t = newNode(y), t->l = merge(x, y->l);
  update(t); return t;
}
void splitByKey(tree t, int k, tree &l, tree &r) { // [−∞, k)[k, +infty)
  if (t == null) l = r = null;
  else if (t->key < k) l = newNode(t), splitByKey(t->r, k, l->r, r), update(l
      );
  else                 r = newNode(t), splitByKey(t->l, k, l, r->l), update(r
      );
}
void splitBySize(tree t, int k, tree &l, tree &r) { // [1, k)[k, +∞)
  static int s; if (t == null) l = r = null;
  else if ((s = t->l->size + 1) < k) l = newNode(t), splitBySize(t->r, k - s,
       l->r, r), update(l);
  else                               r = newNode(t), splitBySize(t->l, k, l,
      r->l), update(r);
}
```

2.4 左偏树

```
tree merge(tree a, tree b) {
  if (a == null) return b;
  if (b == null) return a;
  if (a->key > b->key) swap(a, b);
  a->rc = merge(a->rc, b);
  a->rc->fa = a;
  if (a->lc->dist < a->rc->dist) swap(a->lc, a->rc);
  a->dist = a->rc->dist + 1;
  return a;
}
void erase(tree t) {
  tree x = t->fa, y = merge(t->lc, t->rc);
  if (y != null) y->fa = x;
  if (x == null) root = y;
  else
```

```
  for ((x->lc == t ? x->lc : x->rc) = y; x != null; y = x, x = x->fa) {
    if (x->lc->dist < x->rc->dist) swap(x->lc, x->rc);
    if (x->rc->dist + 1 == x->dist) return;
    x->dist = x->rc->dist + 1;
  }
}
```

2.5 Link-Cut Tree

```
struct node { int rev; node *pre, *ch[2]; } base[MAXN], nil, *null;
typedef node *tree;
#define isRoot(x) (x->pre->ch[0] != x && x->pre->ch[1] != x)
#define isRight(x) (x->pre->ch[1] == x)
inline void MakeRev(tree t) { if (t != null) { t->rev ^= 1; swap(t->ch[0], t
    ->ch[1]); } }
inline void PushDown(tree t) { if (t->rev) { MakeRev(t->ch[0]); MakeRev(t->ch
    [1]); t->rev = 0; } }
inline void Rotate(tree x) {
  tree y = x->pre; PushDown(y); PushDown(x);
  int d = isRight(x);
  if (!isRoot(y)) y->pre->ch[isRight(y)] = x; x->pre = y->pre;
  if ((y->ch[d] = x->ch[!d]) != null) y->ch[d]->pre = y;
  x->ch[!d] = y; y->pre = x; Update(y);
}
inline void Splay(tree x) {
  PushDown(x); for (tree y; !isRoot(x); Rotate(x)) {
    y = x->pre; if (!isRoot(y)) Rotate(isRight(x) != isRight(y) ? x : y);
  } Update(x);
}
inline void Splay(tree x, tree to) {
  PushDown(x); for (tree y; (y = x->pre) != to; Rotate(x)) if (y->pre != to)
    Rotate(isRight(x) != isRight(y) ? x : y);
  Update(x);
}
inline tree Access(tree t) {
  tree last = null; for (; t != null; last = t, t = t->pre) Splay(t),t->ch[1]
      = last, Update(t);
  return last;
}
inline void MakeRoot(tree t) { Access(t); Splay(t); MakeRev(t); }
inline tree FindRoot(tree t) { Access(t); Splay(t); tree last = null;
  for ( ; t != null; last = t, t = t->ch[0]) PushDown(t); Splay(last); return
      last;
}
inline void Join(tree x, tree y) { MakeRoot(y); y->pre = x; }
inline void Cut(tree t) {Access(t); Splay(t); t->ch[0]->pre = null; t->ch[0]
    = null; Update(t);}
inline void Cut(tree x, tree y) {
  tree upper = (Access(x), Access(y));
  if (upper == x) { Splay(x); y->pre = null; x->ch[1] = null; Update(x); }
  else if (upper == y) { Access(x); Splay(y); x->pre = null; y->ch[1] = null;
      Update(y); }
  else assert(0); // impossible to happen
}
inline int Query(tree a, tree b) { // query the cost in path a <-> b, lca inclusive
  Access(a); tree c = Access(b); // c is lca
  int v1 = c->ch[1]->maxCost; Access(a);
  int v2 = c->ch[1]->maxCost;
  return max(max(v1, v2), c->cost);
}
void Init() {
  null = &nil; null->ch[0] = null->ch[1] = null->pre = null; null->rev = 0;
  Rep(i, 1, N) { node &n = base[i]; n.rev = 0; n.pre = n.ch[0] = n.ch[1] =
      null; }
}
```

2.6 K-D Tree Nearest

```
struct Point { int x, y; };
struct Rectangle {
  int lx , rx , ly , ry;
  void set(const Point &p) { lx = rx = p.x; ly = ry = p.y; }
  void merge(const Point &o) {
    lx = min(lx, o.x); rx = max(rx, o.x); ly = min(ly, o.y); ry = max(ry, o.y
        );
  } void merge(const Rectangle &o) {
    lx = min(lx , o.lx); rx = max(rx , o.rx); ly = min(ly , o.ly); ry = max(
        ry , o.ry);
  } LL dist(const Point &p) {
    LL res = 0;
    if (p.x < lx) res += sqr(lx - p.x); else if (p.x > rx) res += sqr(p.x -
        rx);
    if (p.y < ly) res += sqr(ly - p.y); else if (p.y > ry) res += sqr(p.y -
        ry);
    return res;
  }
};
struct Node { int child[2]; Point p; Rectangle rect; };
const int MAX_N = 1111111;
const LL INF = 100000000;
int n, m, tot, root; LL result;
Point a[MAX_N], p; Node tree[MAX_N];
int build(int s, int t, bool d) {
  int k = ++tot, mid = (s + t) >> 1;
  nth_element(a + s, a + mid , a + t, d ? cmpXY : cmpYX);
  tree[k].p = a[mid]; tree[k].rect.set(a[mid]); tree[k].child[0] = tree[k].
      child[1] = 0;
  if (s < mid)
    tree[k].child[0] = build(s, mid , d ^ 1), tree[k].rect.merge(tree[tree[k
        ].child[0]].rect);
  if (mid + 1 < t)
    tree[k].child[1] = build(mid + 1, t, d ^ 1), tree[k].rect.merge(tree[tree
        [k].child[1]].rect);
  return k;
}
int insert(int root, bool d) {
  if (root == 0) {
    tree[++tot].p = p; tree[tot].rect.set(p); tree[tot].child[0] = tree[tot].
        child[1] = 0;
    return tot;
  } tree[root].rect.merge(p);
  if ((d && cmpXY(p, tree[root].p)) || (!d && cmpYX(p, tree[root].p)))
     tree[root].child[0] = insert(tree[root].child[0], d ^ 1);
  else tree[root].child[1] = insert(tree[root].child[1], d ^ 1);
  return root;
}
void query(int k, bool d) {
  if (tree[k].rect.dist(p) >= result) return;
  cMin(result, dist(tree[k].p, p));
```

```
  if ((d && cmpXY(p, tree[k].p)) || (!d && cmpYX(p, tree[k].p))) {
    if (tree[k].child[0]) query(tree[k].child[0], d ^ 1);
    if (tree[k].child[1]) query(tree[k].child[1], d ^ 1);
  } else {
    if (tree[k].child[1]) query(tree[k].child[1], d ^ 1);
    if (tree[k].child[0]) query(tree[k].child[0], d ^ 1);
  }
}
void example(int n) {
  root = tot = 0; scan(a); root = build(0, n, 0); // init, a[0...n-1]
  scan(p); root = insert(root, 0); // insert
  scan(p); result = INF; ans = query(root, 0); // query
}
```

2.7 K-D Tree Farthest

输入 n 个点, 对每个询问 px, py, k, 输出 k 远点的编号

```
struct Point { int x, y, id; };
struct Rectangle {
  int lx, rx, ly, ry;
  void set(const Point &p) { lx = rx = p.x; ly = ry = p.y; }
  void merge(const Rectangle &o) {
    lx = min(lx, o.lx); rx = max(rx, o.rx); ly = min(ly, o.ly); ry = max(ry, o.ry);
  }
  LL dist(const Point &p) { LL res = 0;
    res += max(sqr(rx - p.x), sqr(lx - p.x));
    res += max(sqr(ry - p.y), sqr(ly - p.y));
    return res;
  }
}; struct Node { Point p; Rectangle rect; };
const int MAX_N = 111111;
const LL INF = 1LL << 60;
int n, m;
Point a[MAX_N], b[MAX_N ];
Node tree[MAX_N * 3];
Point p; // p is the query point
pair<LL, int> result[22];
void build(int k, int s, int t, bool d) {
  int mid = (s + t) >> 1;
  nth_element(a + s, a + mid , a + t, d ? cmpX : cmpY);
  tree[k].p = a[mid];
  tree[k].rect.set(a[mid]);
  if (s < mid)
    build(k << 1, s, mid , d ^ 1), tree[k].rect.merge(tree[k << 1]. rect);
  if (mid + 1 < t)
    build(k << 1 | 1, mid + 1, t, d ^ 1), tree[k].rect.merge(tree[k << 1 | 1]. rect);
}
void query(int k, int s, int t, bool d, int kth) {
  if (tree[k].rect.dist(p) < result[kth].first) return;
  pair<LL, int> tmp(dist(tree[k].p, p), -tree[k].p.id);
  for (int i = 1; i <= kth; i++) if (tmp > result[i]) {
    for (int j = kth + 1; j > i; j--) result[j] = result[j - 1]; result[i] = tmp;
    break;
  }
  int mid = (s + t) >> 1;
  if ((d && cmpX(p, tree[k].p)) || (!d && cmpY(p, tree[k].p))) {
    if (mid + 1 < t) query(k << 1 | 1, mid + 1, t, d ^ 1, kth);
    if (s < mid)     query(k << 1, s, mid , d ^ 1, kth);
  } else {
    if (s < mid)     query(k << 1, s, mid , d ^ 1, kth);
    if (mid + 1 < t) query(k << 1 | 1, mid + 1, t, d ^ 1, kth);
  }
}
void example(int n) {
  scan(a); build(1, 0, n, 0); // init, a[0...n-1]
  scan(p, k); // query
  Rep(j, 1, k) result[j].first = -1;
  query(1, 0, n, 0, k); ans = -result[k].second + 1;
}
```

2.8 树链剖分

```
int N, fa[MAXN], dep[MAXN], que[MAXN], size[MAXN], own[MAXN];
int LCA(int x, int y) { if (x == y) return x;
  for ( ; ; x = fa[own[x]]) {
    if (dep[x] < dep[y]) swap(x, y); if (own[x] == own[y]) return y;
    if (dep[own[x]] < dep[own[y]]) swap(x, y);
  } return -1;
}
void Decomposion() {
  static int path[MAXN]; int x, y, a, next, head = 0, tail = 0, cnt; // BFS omitted
  for (int i = 1; i <= N; ++i) if (own[a = que[i]] == -1)
    for (x = a, cnt = 0; ; x = next) { next = -1; own[x] = a; path[++cnt] = x;
      for (edge e(fir[x]); e; e = e->next) if ( (y = e->to) != fa[x] )
        if (next == -1 || size[y] > size[next]) next = y;
      if (next == -1) { tree[a].init(cnt, path); break; }
    }
}
```

3 字符串相关

3.1 Manacher

```
// len[i] : the max length of palindrome whose mid point is (i / 2)
void Manacher(int n, char cs[], int len[]) { // 0-based, len[] must be double sized
  for (int i = 0; i < n + n; ++i) len[i] = 0;
  for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0)) {
    while (i - j >= 0 && i + j + 1 < n * 2 && cs[(i - j) / 2] == cs[(i + j + 1) / 2]) j++;
    len[i] = j; for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; k++)
      len[i + k] = min(len[i - k], j - k);
  }
}
```

3.2 KMP

$next[i] = \max\{len | A[0 \ldots len-1] = A$的第 i 位向前或后的长度为 len 的串$\}$
$ext[i] = \max\{len | A[0 \ldots len-1] = B$的第 i 位向前或后的长度为 len 的串$\}$

```cpp
void KMP(char *a, int la, char *b, int lb, int *next, int *ext) {
  --a; --b; --next; --ext;
  for (int i = 2, j = next[1] = 0; i <= la; i++) {
    while (j && a[j + 1] != a[i]) j = next[j]; if (a[j + 1] == a[i]) ++j;
        next[i] = j;
  } for (int i = 1, j = 0; i <= lb; ++i) {
    while (j && a[j + 1] != b[i]) j = next[j]; if (a[j + 1] == b[i]) ++j; ext
        [i] = j;
    if (j == la) j = next[j];
  }
} void ExKMP(char *a, int la, char *b, int lb, int *next, int *ext) {
  next[0] = la; for (int &j = next[1] = 0; j + 1 < la && a[j] == a[j + 1]; ++
      j);
  for (int i = 2, k = 1; i < la; ++i) {
    int p = k + next[k], l = next[i - k]; if (l < p - i) next[i] = l;
    else for (int &j = next[k = i] = max(0, p - i); i + j < la && a[j] == a[i
        + j]; ++j);
  } for (int &j = ext[0] = 0; j < la && j < lb && a[j] == b[j]; ++j);
  for (int i = 1, k = 0; i < lb; ++i) {
    int p = k + ext[k], l = next[i - k]; if (l < p - i) ext[i] = l;
    else for (int &j = ext[k = i] = max(0, p - i); j < la && i + j < lb && a[
        j] == b[i + j]; ++j);
  }
}
```

3.3 后缀自动机

```cpp
struct node { int len; node *fa, *go[26]; } base[MAXNODE], *top = base, *root
    , *que[MAXNODE];
typedef node *tree;
inline tree newNode(int len) {
  top->len = len; top->fa = NULL; memset(top->go, 0, sizeof(top->go)); return
      top++;
} inline tree newNode(int len, tree fa, tree *go) {
  top->len = len; top->fa = fa; memcpy(top->go, go, sizeof(top->go)); return
      top++;
} void construct(char *A, int N) {
  tree p = root = newNode(0), q, up, fa;
  for (int i = 0; i < N; ++i) {
    int w = A[i] - 'a'; up = p; p = newNode(i + 1);
    for ( ; up && !up->go[w]; up = up->fa) up->go[w] = p;
    if (!up) p->fa = root;
    else { q = up->go[w];
      if (up->len + 1 == q->len) p->fa = q;
      else { fa = newNode(up->len + 1, q->fa, q->go);
        for (p->fa = q->fa = fa; up && up->go[w] == q; up = up->fa) up->go[w]
            = fa;
      }
    }
  } static int cnt[MAXLEN]; memset(cnt, 0, sizeof(int) * (N + 1));
  for (tree i(base); i != top; ++i) ++cnt[i->len];
  Rep(i, 1, N) cnt[i] += cnt[i - 1];
  for (tree i(base); i != top; ++i) Q[ cnt[i->len]-- ] = i;
}
```

3.4 后缀数组

待排序的字符串放在 $r[0 \dots n-1]$ 中, 最大值小于 m.
$r[0 \dots n-2] > 0$, $r[n-1] = 0$.
结果放在 $sa[0 \dots n-1]$.

```cpp
namespace SuffixArrayDoubling {
  int wa[MAXN], wb[MAXN], wv[MAXN], ws[MAXN];
  int cmp(int *r, int a, int b, int l) { return r[a] == r[b] && r[a + l] == r
      [b + l]; }
  void da(int *r, int *sa, int n, int m) {
    int i, j, p, *x = wa, *y = wb, *t;
    for (i = 0; i < m; i++) ws[i] = 0;
    for (i = 0; i < n; i++) ws[x[i] = r[i]]++;
    for (i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (i = n - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
    for (j = 1, p = 1; p < n; j *= 2, m = p) {
      for (p = 0, i = n - j; i < n; i++) y[p++] = i;
      for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
      for (i = 0; i < n; i++) wv[i] = x[y[i]];
      for (i = 0; i < m; i++) ws[i] = 0;
      for (i = 0; i < n; i++) ws[wv[i]]++;
      for (i = 1; i < m; i++) ws[i] += ws[i - 1];
      for (i = n - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
      for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
        x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
}}}
namespace CalcHeight {
  int rank[MAXN], height[MAXN];
  void calheight(int *r, int *sa, int n) {
    int i, j, k = 0; for (i = 1; i <= n; i++) rank[sa[i]] = i;
    for (i = 0; i < n; height[rank[i++]] = k)
      for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);
}}
```

3.5 环串最小表示

```cpp
int minimalRepresentation(int N, char *s) { // s must be double-sized and 0-
    based
  int i, j, k, l; for (i = 0; i < N; ++i) s[i + N] = s[i]; s[N + N] = 0;
  for (i = 0, j = 1; j < N; ) {
    for (k = 0; k < N && s[i + k] == s[j + k]; ++k);
    if (k >= N) break; if (s[i + k] < s[j + k]) j += k + 1;
    else l = i + k, i = j, j = max(l, j) + 1;
  } return i; // [i, i + N) is the minimal representation
}
```

4 图论

4.1 带花树

```cpp
namespace Blossom {
  int n, head, tail, S, T, lca;
  int match[MAXN], Q[MAXN], pred[MAXN], label[MAXN], inq[MAXN], inb[MAXN];
  vector<int> link[MAXN];
  inline void push(int x) { Q[tail++] = x; inq[x] = true; }
  int findCommonAncestor(int x, int y) {
    static bool inPath[MAXN]; for (int i = 0; i < n; ++i) inPath[i] = 0;
    for ( ; ; x = pred[ match[x] ]) { x = label[x]; inPath[x] = true; if (x
        == S) break; }
```

```
    for ( ; ; y = pred[ match[y] ]) { y = label[y]; if (inPath[y]) break; } return y;
  }
  void resetTrace(int x, int lca) {
    while (label[x] != lca) { int y = match[x]; inb[ label[x] ] = inb[ label[y] ] = true;
      x = pred[y]; if (label[x] != lca) pred[x] = y; }}
  void blossomContract(int x, int y) {
    lca = findCommonAncestor(x, y);
    Foru(i, 0, n) inb[i] = 0; resetTrace(x, lca); resetTrace(y, lca);
    if (label[x] != lca) pred[x] = y; if (label[y] != lca) pred[y] = x;
    Foru(i, 0, n) if (inb[ label[i] ]) { label[i] = lca; if (!inq[i]) push(i); }
  }
  bool findAugmentingPath() {
    Foru(i, 0, n) pred[i] = -1, label[i] = i, inq[i] = 0;
    int x, y, z; head = tail = 0;
    for (push(S); head < tail; ) for (int i = (int)link[x = Q[head++]].size() - 1; i >= 0; --i) {
      y = link[x][i]; if (label[x] == label[y] || x == match[y]) continue;
      if (y == S || (match[y] >= 0 && pred[ match[y] ] >= 0)) blossomContract(x, y);
      else if (pred[y] == -1) {
        pred[y] = x; if (match[y] >= 0) push(match[y]);
        else {
          for (x = y; x >= 0; x = z) {
          y = pred[x], z = match[y]; match[x] = y, match[y] = x;
        } return true; }}} return false;
  }
  int findMaxMatching() {
    int ans = 0; Foru(i, 0, n) match[i] = -1;
    for (S = 0; S < n; ++S) if (match[S] == -1) if (findAugmentingPath()) ++ans;
    return ans;
  }
}
```

4.2 最大流

```
namespace Maxflow {
  int h[MAXNODE], vh[MAXNODE], S, T, Ncnt; edge cur[MAXNODE], pe[MAXNODE];
  void init(int _S, int _T, int _Ncnt) { S = _S; T = _T; Ncnt = _Ncnt; }
  int maxflow() {
    static int Q[MAXNODE]; int x, y, augc, flow = 0, head = 0, tail = 0; edge e;
    Rep(i, 0, Ncnt) cur[i] = fir[i]; Rep(i, 0, Ncnt) h[i] = INF; Rep(i, 0, Ncnt) vh[i] = 0;
    for (Q[++tail] = T, h[T] = 0; head < tail; ) {
      x = Q[++head]; ++vh[ h[x] ];
      for (e = fir[x]; e; e = e->next) if (e->op->c)
        if (h[y = e->to] >= INF) h[y] = h[x] + 1, Q[++tail] = y;
    } for (x = S; h[S] < Ncnt; ) {
      for (e = cur[x]; e; e = e->next) if (e->c)
        if (h[y = e->to] + 1 == h[x]) { cur[x] = pe[y] = e; x = y; break; }
      if (!e) {
        if (--vh[ h[x] ] == 0) break; h[x] = Ncnt; cur[x] = NULL;
        for (e = fir[x]; e; e = e->next) if (e->c)
          if ( cMin( h[x], h[e->to] + 1 ) ) cur[x] = e;
        ++vh[ h[x] ];
        if (x != S) x = pe[x]->op->to;
      } else if (x == T) { augc = INF;
        for (x = T; x != S; x = pe[x]->op->to) cMin(augc, pe[x]->c);
        for (x = T; x != S; x = pe[x]->op->to) {
          pe[x]->c -= augc; pe[x]->op->c += augc;
        } flow += augc;
      }
    } return flow;
  }
}
```

4.3 KM

```
int N, Tcnt, w[MAXN][MAXN], slack[MAXN];
int lx[MAXN], linkx[MAXN], visy[MAXN], ly[MAXN], linky[MAXN], visx[MAXN]; // 初值全为0
bool DFS(int x) { visx[x] = Tcnt;
  Rep(y, 1, N) if(visy[y] != Tcnt) { int t = lx[x] + ly[y] - w[x][y];
    if (t == 0) { visy[y] = Tcnt;
      if (!linky[y] || DFS(linky[y])) { linkx[x] = y; linky[y] = x; return true; }
    } else cMin(slack[y], t);
  } return false;
} void KM() {
  Tcnt = 0; Rep(x, 1, N) Rep(y, 1, N) cMax(lx[x], w[x][y]);
  Rep(S, 1, N) { Rep(i, 1, N) slack[i] = INF;
    for (++Tcnt; !DFS(S); ++Tcnt) { int d = INF;
      Rep(y, 1, N) if(visy[y] != Tcnt) cMin(d, slack[y]);
      Rep(x, 1, N) if(visx[x] == Tcnt) lx[x] -= d;
      Rep(y, 1, N) if(visy[y] == Tcnt) ly[y] += d; else slack[y] -= d;
    }
  }
}
```

4.4 2-SAT 与 Kosaraju

注意 Kosaraju 需要建反图

```
namespace SCC {
  int code[MAXN * 2], seq[MAXN * 2], sCnt;
  void DFS_1(int x) { code[x] = 1;
    for (edge e(fir[x]); e; e = e->next) if (code[e->to] == -1) DFS_1(e->to);
    seq[++sCnt] = x;
  } void DFS_2(int x) { code[x] = sCnt;
    for (edge e(fir2[x]); e; e = e->next) if (code[e->to] == -1) DFS_2(e->to); }
  void SCC(int N) {
    sCnt = 0; for (int i = 1; i <= N; ++i) code[i] = -1;
    for (int i = 1; i <= N; ++i) if (code[i] == -1) DFS_1(i);
    sCnt = 0; for (int i = 1; i <= N; ++i) code[i] = -1;
    for (int i = N; i >= 1; --i) if (code[seq[i]] == -1) {
      ++sCnt; DFS_2(seq[i]); }
  }
}// true - 2i - 1
// false - 2i
bool TwoSat() { SCC::SCC(N + N);
  // if code[2i - 1] = code[2i]: no solution
  // if code[2i - 1] > code[2i]: i selected. else i not selected
```

```
}
```

4.5 全局最小割 Stoer-Wagner

```
int minCut(int N, int G[MAXN][MAXN]) { // 0-based
  static int weight[MAXN], used[MAXN]; int ans = INT_MAX;
  while (N > 1) {
    for (int i = 0; i < N; ++i) used[i] = false; used[0] = true;
    for (int i = 0; i < N; ++i) weight[i] = G[i][0];
    int S = -1, T = 0;
    for (int _r = 2; _r <= N; ++_r) { // N - 1 selections
      int x = -1;
      for (int i = 0; i < N; ++i) if (!used[i])
        if (x == -1 || weight[i] > weight[x]) x = i;
      for (int i = 0; i < N; ++i) weight[i] += G[x][i];
      S = T; T = x; used[x] = true;
    } ans = min(ans, weight[T]);
    for (int i = 0; i < N; ++i) G[i][S] += G[i][T], G[S][i] += G[i][T];
    G[S][S] = 0; --N;
    for (int i = 0; i <= N; ++i) swap(G[i][T], G[i][N]);
    for (int i = 0; i < N; ++i) swap(G[T][i], G[N][i]);
  } return ans;
}
```

4.6 欧拉路

```
vector<int> eulerianWalk(int N, int S) {
  static int res[MAXM], stack[MAXN]; static edge cur[MAXN];
  int rcnt = 0, top = 0, x; for (int i = 1; i <= N; ++i) cur[i] = fir[i];
  for (stack[top++] = S; top; ) {
    for (x = stack[--top]; ; ) {
      edge &e = cur[x]; if (e == NULL) break;
      stack[top++] = x; x = e->to; e = e->next;
      // 对于无向图需要删掉反向边
    } res[rcnt++] = x;
  } reverse(res, res + rcnt); return vector<int>(res, res + rcnt);
}
```

4.7 最大团搜索

```
namespace MaxClique { // 1-based
  int g[MAXN][MAXN], len[MAXN], list[MAXN][MAXN], mc[MAXN], ans, found;
  void DFS(int size) {
    if (len[size] == 0) { if (size > ans) ans = size, found = true; return; }
    for (int k = 0; k < len[size] && !found; ++k) {
      if (size + len[size] - k <= ans) break;
      int i = list[size][k]; if (size + mc[i] <= ans) break;
      for (int j = k + 1, len[size + 1] = 0; j < len[size]; ++j) if (g[i][
          list[size][j]])
        list[size + 1][len[size + 1]++] = list[size][j];
      DFS(size + 1);
    }
  }
  int work(int n) {
    mc[n] = ans = 1; for (int i = n - 1; i; --i) { found = false; len[1] = 0;
      for (int j = i + 1; j <= n; ++j) if (g[i][j]) list[1][len[1]++] = j;
      DFS(1); mc[i] = ans;
    } return ans;
  }
}
```

4.8 最小树形图

```
namespace EdmondsAlgorithm { // O(ElogE + V^2) !!! 0-based !!!
  struct enode { int from, c, key, delta, dep; enode *ch[2], *next;
  } ebase[maxm], *etop, *fir[maxn], nil, *null, *inEdge[maxn], *chs[maxn];
  typedef enode *edge; typedef enode *tree;
  int n, m, setFa[maxn], deg[maxn], que[maxn];
  inline void pushDown(tree x) { if (x->delta) {
    x->ch[0]->key += x->delta; x->ch[0]->delta += x->delta;
    x->ch[1]->key += x->delta; x->ch[1]->delta += x->delta; x->delta = 0;
  }}
  tree merge(tree x, tree y) {
    if (x == null) return y; if (y == null) return x;
    if (x->key > y->key) swap(x, y); pushDown(x); x->ch[1] = merge(x->ch[1],
        y);
    if (x->ch[0]->dep < x->ch[1]->dep) swap(x->ch[0], x->ch[1]);
    x->dep = x->ch[1]->dep + 1; return x;
  }
  void addEdge(int u, int v, int w) {
    etop->from = u; etop->c = etop->key = w; etop->delta = etop->dep = 0;
    etop->next = fir[v]; etop->ch[0] = etop->ch[1] = null;
    fir[v] = etop; inEdge[v] = merge(inEdge[v], etop++);
  }
  void deleteMin(tree &r) { pushDown(r); r = merge(r->ch[0], r->ch[1]); }
  int findSet(int x) { return setFa[x] == x ? x : setFa[x] = findSet(setFa[x
      ]); }
  void clear(int V, int E) {
    null = &nil; null->ch[0] = null->ch[1] = null; null->dep = -1;
    n = V; m = E; etop = ebase; Foru(i, 0, V) fir[i] = NULL; Foru(i, 0, V)
        inEdge[i] = null;
  }
  int solve(int root) { int res = 0, head, tail;
    for (int i = 0; i < n; ++i) setFa[i] = i;
    for ( ; ; ) { memset(deg, 0, sizeof(int) * n); chs[root] = inEdge[root];
      for (int i = 0; i < n; ++i) if (i != root && setFa[i] == i) {
        while (findSet(inEdge[i]->from) == findSet(i)) deleteMin(inEdge[i]);
        ++deg[ findSet((chs[i] = inEdge[i])->from) ];
      }
      for (int i = head = tail = 0; i < n; ++i)
        if (i != root && setFa[i] == i && deg[i] == 0) que[tail++] = i;
      while (head < tail) {
        int x = findSet(chs[que[head++]]->from);
        if (--deg[x] == 0) que[tail++] = x;
      } bool found = false;
      for (int i = 0; i < n; ++i) if (i != root && setFa[i] == i && deg[i] >
          0) {
        int j = i; tree temp = null; found = true;
        do {setFa[j = findSet(chs[j]->from)] = i;
          deleteMin(inEdge[j]); res += chs[j]->key;
          inEdge[j]->key -= chs[j]->key; inEdge[j]->delta -= chs[j]->key;
          temp = merge(temp, inEdge[j]);
        } while (j != i); inEdge[i] = temp;
```

```cpp
      } if (!found) break;
    } for (int i = 0; i < n; ++i) if (i != root && setFa[i] == i) res += chs[i]->key;
    return res;
  }
}
namespace ChuLiu { // O(V ^ 3) !!! 1-based !!!
  int n, used[maxn], pass[maxn], eg[maxn], more, que[maxn], g[maxn][maxn];
  void combine(int id, int &sum) { int tot = 0, from, i, j, k;
    for ( ; id != 0 && !pass[id]; id = eg[id]) que[tot++] = id, pass[id] = 1;
    for (from = 0; from < tot && que[from] != id; from++);
    if (from == tot) return; more = 1;
    for (i = from; i < tot; i++) {
      sum += g[eg[que[i]]][que[i]]; if (i == from) continue;
      for (j = used[que[i]] = 1; j <= n; j++) if (!used[j])
        if (g[que[i]][j] < g[id][j]) g[id][j] = g[que[i]][j];
    }
    for (i = 1; i <= n; i++) if (!used[i] && i != id)
      for (j = from; j < tot; j++) {
        k = que[j]; if (g[i][id] > g[i][k] - g[eg[k]][k])
        g[i][id] = g[i][k] - g[eg[k]][k];
      }
  }
  void clear(int V) { n = V; Rep(i, 1, V) Rep(j, 1, V) g[i][j] = inf; }
  int solve(int root) {
    int i, j, k, sum = 0; memset(used, 0, sizeof(int) * (n + 1));
    for (more = 1; more; ) {
      more = 0; memset(eg, 0, sizeof(int) * (n + 1));
      for (i = 1; i <= n; i++) if (!used[i] && i != root) {
        for (j = 1, k = 0; j <= n; j++) if (!used[j] && i != j)
          if (k == 0 || g[j][i] < g[k][i]) k = j;
        eg[i] = k;
      } memset(pass, 0, sizeof(int) * (n + 1));
      for (i = 1; i <= n; i++) if (!used[i] && !pass[i] && i != root)
        combine(i, sum);
    } for (i = 1; i <= n; i++) if (!used[i] && i != root) sum += g[eg[i]][i];
    return sum;
  }
}
```

4.9 离线动态最小生成树

$O(Q\log^2 Q)$. $(qx[i], qy[i])$ 表示将编号为 $qx[i]$ 的边的权值改为 $qy[i]$, 删除一条边相当于将其权值改为 ∞, 加入一条边相当于将其权值从 ∞ 变成某个值.

```cpp
const int maxn = 100000 + 5;
const int maxm = 1000000 + 5;
const int maxq = 1000000 + 5;
const int qsize = maxm + 3 * maxq;
int n, m, Q, x[qsize], y[qsize], z[qsize], qx[maxq], qy[maxq], a[maxn], *tz;
int kx[maxn], ky[maxn], kt, vd[maxn], id[maxm], app[maxm];
bool extra[maxm];
void init() {
  scanf("%d%d", &n, &m); for (int i = 0; i < m; i++) scanf("%d%d%d", x + i, y + i, z + i);
  scanf("%d", &Q); for (int i = 0; i < Q; i++) { scanf("%d%d", qx + i, qy + i); qx[i]--; }
}
int find(int x) {
  int root = x, next; while (a[root]) root = a[root];
  while ((next = a[x]) != 0) a[x] = root, x = next; return root;
}
inline bool cmp(const int &a, const int &b) { return tz[a] < tz[b]; }
void solve(int *qx, int *qy, int Q, int n, int *x, int *y, int *z, int m,
    long long ans) {
  int ri, rj;
  if (Q == 1) {
    for (int i = 1; i <= n; i++) a[i] = 0; z[qx[0]] = qy[0];
    for (int i = 0; i < m; i++) id[i] = i;
    tz = z; sort(id, id + m, cmp);
    for (int i = 0; i < m; i++) {
      ri = find(x[id[i]]); rj = find(y[id[i]]);
      if (ri != rj) ans += z[id[i]], a[ri] = rj;
    } printf("%I64d\n", ans);
    return;
  } int tm = kt = 0, n2 = 0, m2 = 0;
  for (int i = 1; i <= n; i++) a[i] = 0;
  for (int i = 0; i < Q; i++) {
    ri = find(x[qx[i]]); rj = find(y[qx[i]]); if (ri != rj) a[ri] = rj;
  }
  for (int i = 0; i < m; i++) extra[i] = true;
  for (int i = 0; i < Q; i++) extra[qx[i]] = false;
  for (int i = 0; i < m; i++) if (extra[i]) id[tm++] = i;
  tz = z; sort(id, id + tm, cmp);
  for (int i = 0; i < tm; i++) {
    ri = find(x[id[i]]); rj = find(y[id[i]]);
    if (ri != rj)
      a[ri] = rj, ans += z[id[i]], kx[kt] = x[id[i]], ky[kt] = y[id[i]], kt
          ++;
  }
  for (int i = 1; i <= n; i++) a[i] = 0;
  for (int i = 0; i < kt; i++) a[find(kx[i])] = find(ky[i]);
  for (int i = 1; i <= n; i++) if (a[i] == 0) vd[i] = ++n2;
  for (int i = 1; i <= n; i++) if (a[i] != 0) vd[i] = vd[find(i)];
  int *Nx = x + m, *Ny = y + m, *Nz = z + m;
  for (int i = 0; i < m; i++) app[i] = -1;
  for (int i = 0; i < Q; i++)
    if (app[qx[i]] == -1)
      Nx[m2] = vd[x[qx[i]]], Ny[m2] = vd[y[qx[i]]], Nz[m2] = z[qx[i]], app[qx
          [i]] = m2, m2++;
  for (int i = 0; i < Q; i++) {
    z[qx[i]] = qy[i];
    qx[i] = app[qx[i]];
  }
  for (int i = 1; i <= n2; i++) a[i] = 0;
  for (int i = 0; i < tm; i++) {
    ri = find(vd[x[id[i]]]); rj = find(vd[y[id[i]]]);
    if (ri != rj)
      a[ri] = rj, Nx[m2] = vd[x[id[i]]], Ny[m2] = vd[y[id[i]]], Nz[m2] = z[id
          [i]], m2++;
  }
  int mid = Q / 2;
  solve(qx, qy, mid, n2, Nx, Ny, Nz, m2, ans);
  solve(qx + mid, qy + mid, Q - mid, n2, Nx, Ny, Nz, m2, ans);
}
void work() { if (Q) solve(qx, qy, Q, n, x, y, z, m, 0); }
int main() { init(); work(); return 0; }
```

4.10 小知识

- 平面图: 一定存在一个度小于等于 5 的点. $E \le 3V - 6$. 欧拉公式: $V + F - E = 1 +$ 连通块数
- 图连通度:
  1. $k-$ 连通 (k-*connected*): 对于任意一对结点都至少存在结点各不相同的 k 条路
  2. 点连通度 (*vertex connectivity*): 把图变成非连通图所需删除的最少点数
  3. Whitney 定理: 一个图是 $k-$ 连通的当且仅当它的点连通度至少为 k
- Lindstroem-Gessel-Viennot Lemma: 给定一个图的 n 个起点和 n 个终点, 令 $A_{ij} =$ 第 i 个起点到第 j 个终点的路径条数, 则从起点到终点的不相交路径条数为 $det(A)$
- 欧拉回路与树形图的联系: 对于出度等于入度的连通图 $s(G) = t_i(G) \prod_{j=1}^{n} (d^+(v_j) - 1)!$
- 密度子图: 给定无向图, 选取点集及其导出子图, 最大化 $W_e + P_v$ (点权可负).
  - $(S, u) = U$, $(u, T) = U - 2P_u - D_u$, $(u, v) = (v, u) = W_e$
  - ans $= \frac{Un - C[S,T]}{2}$, 解集为 $S - \{s\}$
- 最大权闭合图: 选 a 则 a 的后继必须被选
  - $P_u > 0$, $(S, u) = P_u$, $P_u < 0$, $(u, T) = -P_u$
  - ans $= \sum_{P_u > 0} P_u - C[S, T]$, 解集为 $S - \{s\}$
- 判定边是否属于最小割:
  - 可能属于最小割: (u, v) 不属于同一 SCC
  - 一定在所有最小割中: (u, v) 不属于同一 SCC, 且 S, u 在同一 SCC, u, T 在同一 SCC

5 数学

5.1 单纯形 Cpp

max $\{cx | Ax \le b, x \ge 0\}$

```
const int MAXN = 11000, MAXM = 1100;
// here MAXN is the MAX number of conditions, MAXM is the MAX number of vars

int avali[MAXM], avacnt;
double A[MAXN][MAXM];
double b[MAXN], c[MAXM];
double* simplex(int n, int m) {
// here n is the number of conditions, m is the number of vars
  m++;
  int r = n, s = m - 1;
  static double D[MAXN + 2][MAXM + 1];
  static int ix[MAXN + MAXM];
  for (int i = 0; i < n + m; i++) ix[i] = i;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m - 1; j++) D[i][j] = -A[i][j];
    D[i][m - 1] = 1;
    D[i][m] = b[i];
    if (D[r][m] > D[i][m]) r = i;
  }
  for (int j = 0; j < m - 1; j++) D[n][j] = c[j];
  D[n + 1][m - 1] = -1;
  for (double d; ; ) {
    if (r < n) {
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
      D[r][s] = 1.0 / D[r][s];
      for (int j = 0; j <= m; j++) if (j != s) D[r][j] *= -D[r][s];
      avacnt = 0;
      for (int i = 0; i <= m; ++i)
        if(fabs(D[r][i]) > EPS)
          avali[avacnt++] = i;
      for (int i = 0; i <= n + 1; i++) if (i != r) {
        if(fabs(D[i][s]) < EPS) continue;
        double *cur1 = D[i], *cur2 = D[r], tmp = D[i][s];
        //for (int j = 0; j <= m; j++) if (j != s) cur1[j] += cur2[j] * tmp;
        for(int j = 0; j < avacnt; ++j) if(avali[j] != s) cur1[avali[j]] +=
            cur2[avali[j]] * tmp;
        D[i][s] *= D[r][s];
      }
    }
    r = -1; s = -1;
    for (int j = 0; j < m; j++) if (s < 0 || ix[s] > ix[j]) {
      if (D[n + 1][j] > EPS || D[n + 1][j] > -EPS && D[n][j] > EPS) s = j;
    }
    if (s < 0) break;
    for (int i = 0; i < n; i++) if (D[i][s] < -EPS) {
      if (r < 0 || (d = D[r][m] / D[r][s] - D[i][m] / D[i][s]) < -EPS
          || d < EPS && ix[r + m] > ix[i + m])
        r = i;
    }
    if (r < 0) return null; // 非有界
  }
  if (D[n + 1][m] < -EPS) return null; // 无法执行
  static double x[MAXM - 1];
  for (int i = m; i < n + m; i++) if (ix[i] < m - 1) x[ix[i]] = D[i - m][m];
  return x; // 值为 D[n][m]
}
```

5.2 FFT

```
namespace FFT {
  #define mul(a, b) (Complex(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x))
  struct Complex {}; // something omitted
  void FFT(Complex P[], int n, int oper) {
    for (int i = 1, j = 0; i < n - 1; i++) {
      for (int s = n; j ^= s >>= 1, ~j & s; );
      if (i < j) swap(P[i], P[j]);
    }
    for (int d = 0; (1 << d) < n; d++) {
      int m = 1 << d, m2 = m * 2;
      double p0 = PI / m * oper;
      Complex unit_p0(cos(p0), sin(p0));
      for (int i = 0; i < n; i += m2) {
        Complex unit(1.0, 0.0);
        for (int j = 0; j < m; j++) {
          Complex &P1 = P[i + j + m], &P2 = P[i + j];
          Complex t = mul(unit, P1);
          P1 = Complex(P2.x - t.x, P2.y - t.y);
          P2 = Complex(P2.x + t.x, P2.y - t.y);
          unit = mul(unit, unit_p0);
```

```cpp
  }}}}
  vector<int> doFFT(const vector<int> &a, const vector<int> &b) {
    vector<int> ret(max(0, (int) a.size() + (int) b.size() - 1), 0);
    static Complex A[MAXB], B[MAXB], C[MAXB];
    int len = 1; while (len < (int)ret.size()) len *= 2;
    for (int i = 0; i < len; i++) A[i] = i < (int)a.size() ? a[i] : 0;
    for (int i = 0; i < len; i++) B[i] = i < (int)b.size() ? b[i] : 0;
    FFT(A, len, 1); FFT(B, len, 1);
    for (int i = 0; i < len; i++) C[i] = mul(A[i], B[i]);
    FFT(C, len, -1);
    for (int i = 0; i < (int)ret.size(); i++)
      ret[i] = (int) (C[i].x / len + 0.5);
    return ret;
  }
}
```

5.3 整数 FFT

```cpp
namespace FFT {
//  替代方案: 23068673(= 11 * 2^21 + 1), 原根为 3
  const int MOD = 786433, PRIMITIVE_ROOT = 10; // 3 * 2^18 + 1
  const int MAXB = 1 << 20;
  int getMod(int downLimit) { // 或者现场自己找一个 MOD
    for (int c = 3; ; ++c) { int t = (c << 21) | 1;
      if (t >= downLimit && isPrime(t)) return t;
  }}
  int modInv(int a) { return a <= 1 ? a : (long long) (MOD - MOD / a) * modInv(MOD % a) % MOD; }
  void NTT(int P[], int n, int oper) {
    for (int i = 1, j = 0; i < n - 1; i++) {
      for (int s = n; j ^= s >>= 1, ~j & s;);
      if (i < j) swap(P[i], P[j]);
    }
    for (int d = 0; (1 << d) < n; d++) {
      int m = 1 << d, m2 = m * 2;
      long long unit_p0 = powMod(PRIMITIVE_ROOT, (MOD - 1) / m2);
      if (oper < 0) unit_p0 = modInv(unit_p0);
      for (int i = 0; i < n; i += m2) {
        long long unit = 1;
        for (int j = 0; j < m; j++) {
          int &P1 = P[i + j + m], &P2 = P[i + j];
          int t = unit * P1 % MOD;
          P1 = (P2 - t + MOD) % MOD; P2 = (P2 + t) % MOD;
          unit = unit * unit_p0 % MOD;
  }}}}
  vector<int> mul(const vector<int> &a, const vector<int> &b) {
    vector<int> ret(max(0, (int) a.size() + (int) b.size() - 1), 0);
    static int A[MAXB], B[MAXB], C[MAXB];
    int len = 1; while (len < (int)ret.size()) len <<= 1;
    for (int i = 0; i < len; i++) A[i] = i < (int)a.size() ? a[i] : 0;
    for (int i = 0; i < len; i++) B[i] = i < (int)b.size() ? b[i] : 0;
    NTT(A, len, 1); NTT(B, len, 1);
    for (int i = 0; i < len; i++) C[i] = (long long) A[i] * B[i] % MOD;
    NTT(C, len, -1); for (int i = 0, inv = modInv(len); i < (int)ret.size(); i++) ret[i] = (long long) C[i] * inv % MOD;
    return ret;
  }
}
```

5.4 扩展欧几里得

$ax + by = g = gcd(x, y)$

```cpp
void exgcd(LL x, LL y, LL &a0, LL &b0, LL &g) {
  LL a1 = b0 = 0, b1 = a0 = 1, t;
  while (y != 0) {
    t = a0 - x / y * a1, a0 = a1, a1 = t;
    t = b0 - x / y * b1, b0 = b1, b1 = t;
    t = x % y, x = y, y = t;
  } if (x < 0) a0 = -a0, b0 = -b0, x = -x;
  g = x;
}
```

5.5 线性同余方程

- 中国剩余定理: 设 $m_1, m_2, \cdots, m_k$ 两两互素, 则同余方程组 $x \equiv a_i \pmod{m_i}$for $i = 1, 2, \cdots, k$ 在 $[0, M = m_1 m_2 \cdots m_k)$ 内有唯一解. 记 $M_i = M/m_i$, 找出 p_i 使得 $M_i p_i \equiv 1 \pmod{m_i}$, 记 $e_i = M_i p_i$, 则 $x \equiv e_1 a_1 + e_2 a_2 + \cdots + e_k a_k \pmod{M}$

- 多变元线性同余方程组: 方程的形式为 $a_1x_1 + a_2x_2 + \cdots + a_nx_n + b \equiv 0 \pmod{m}$, 令 $d = (a_1, a_2, \cdots, a_n, m)$, 有解的充要条件是 $d|b$, 解的个数为 $m^{n-1}d$

5.6 Miller-Rabin 素性测试

```cpp
bool test(LL n, int base) {
  LL m = n - 1, ret = 0; int s = 0;
  for ( ; m % 2 == 0; ++s) m >>= 1; ret = pow_mod(base, m, n);
  if (ret == 1 || ret == n - 1) return true;
  for (--s; s >= 0; --s) {
    ret = multiply_mod(ret, ret, n); if (ret == n - 1) return true;
  } return false;
}
LL special[7] = {
  1373653LL,          25326001LL,
  3215031751LL,       25000000000LL,
  2152302898747LL,    3474749660383LL, 341550071728321LL};
/*
 * n < 2047                          test[] = {2}
 * n < 1,373,653                     test[] = {2, 3}
 * n < 9,080,191                     test[] = {31, 73}
 * n < 25,326,001                    test[] = {2, 3, 5}
 * n < 4,759,123,141                 test[] = {2, 7, 61}
 * n < 1,122,004,669,633             test[] = {2, 13, 23, 1662803}
 * n < 2,152,302,898,747             test[] = {2, 3, 5, 7, 11}
 * n < 3,474,749,660,383             test[] = {2, 3, 5, 7, 11, 13}
 * n < 341,550,071,728,321           test[] = {2, 3, 5, 7, 11, 13, 17}
 * n < 3,825,123,056,546,413,051     test[] = {2, 3, 5, 7, 11, 13, 17, 19, 23}
 */
bool is_prime(LL n) {
  if (n < 2) return false;
  if (n < 4) return true;
  if (!test(n, 2) || !test(n, 3)) return false;
  if (n < special[0]) return true;
  if (!test(n, 5)) return false;
  if (n < special[1]) return true;
  if (!test(n, 7)) return false;
  if (n == special[2]) return false;
  if (n < special[3]) return true;
```

```
  if (!test(n, 11)) return false;
  if (n < special[4]) return true;
  if (!test(n, 13)) return false;
  if (n < special[5]) return true;
  if (!test(n, 17)) return false;
  if (n < special[6]) return true;
  return test(n, 19) && test(n, 23) && test(n, 29) && test(n, 31) && test(n,
      37);
}
```

5.7 PollardRho

```
LL pollardRho(LL n, LL seed) {
  LL x, y, head = 1, tail = 2; x = y = random() % (n - 1) + 1;
  for ( ; ; ) {
    x = addMod(multiplyMod(x, x, n), seed, n);
    if (x == y) return n; LL d = gcd(myAbs(x - y), n);
    if (1 < d && d < n) return d;
    if (++head == tail) y = x, tail <<= 1;
}} vector<LL> divisors;
void factorize(LL n) { // 需要保证 n > 1
  if (isPrime(n)) divisors.push_back(n);
  else { LL d = n;
    while (d >= n) d = pollardRho(n, random() % (n - 1) + 1);
    factorize(n / d); factorize(d);
}}
```

5.8 多项式求根

```
const double error = 1e-12;
const double infi = 1e+12;
int n; double a[10], x[10];
double f(double a[], int n, double x) {
  double tmp = 1, sum = 0;
  for (int i = 0; i <= n; i++) sum = sum + a[i] * tmp, tmp = tmp * x;
  return sum;
}
double binary(double l, double r, double a[], int n) {
  int sl = sign(f(a, n, l)), sr = sign(f(a, n, r));
  if (sl == 0) return l; if (sr == 0) return r;
  if (sl * sr > 0) return infi;
  while (r - l > error) {
    double mid = (l + r) / 2;
    int ss = sign(f(a, n, mid));
    if (ss == 0) return mid;
    if (ss * sl > 0) l = mid; else r = mid;
  } return l;
}
void solve(int n, double a[], double x[], int &nx) {
  if (n == 1) { x[1] = -a[0] / a[1]; nx = 1; return; }
  double da[10], dx[10]; int ndx;
  for (int i = n; i >= 1; i--) da[i - 1] = a[i] * i;
  solve(n - 1, da, dx, ndx); nx = 0;
  if (ndx == 0) {
    double tmp = binary(-infi, infi, a, n);
    if (tmp < infi) x[++nx] = tmp; return;
  } double tmp = binary(-infi, dx[1], a, n);
  if (tmp < infi) x[++nx] = tmp;
  for (int i = 1; i <= ndx - 1; i++) {
    tmp = binary(dx[i], dx[i + 1], a, n);
    if (tmp < infi) x[++nx] = tmp;
  } tmp = binary(dx[ndx], infi, a, n);
  if (tmp < infi) x[++nx] = tmp;
}
int main() {
  scanf("%d", &n);
  for (int i = n; i >= 0; i--) scanf("%lf", &a[i]);
  int nx; solve(n, a, x, nx);
  for (int i = 1; i <= nx; i++) printf("%0.6f\n", x[i]);
  return 0;
}
```

5.9 线性递推

for $a_{i+n} = (\sum_{i=0}^{n-1} k_j a_{i+j}) + d$, $a_m = (\sum_{i=0}^{n-1} c_i a_i) + c_n d$

```
vector<int> recFormula(int n, int k[], int m) {
  vector<int> c(n + 1, 0);
  if (m < n) c[m] = 1;
  else {
    static int a[MAX_K * 2 + 1];
    vector<int> b = recFormula(n, k, m >> 1);
    for (int i = 0; i < n + n; ++i) a[i] = 0;
    int s = m & 1;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) a[i + j + s] += b[i] * b[j];
      c[n] += b[i];
    } c[n] = (c[n] + 1) * b[n];
    for (int i = n * 2 - 1; i >= n; i--) {
      int add = a[i]; if (add == 0) continue;
      for (int j = 0; j < n; j++) a[i - n + j] += k[j] * add;
      c[n] += add;
    } for (int i = 0; i < n; ++i) c[i] = a[i];
  } return c;
}
```

5.10 原根

原根 g: g 是模 n 简化剩余系构成的乘法群的生成元. 模 n 有原根的充要条件是 $n = 2, 4, p^n, 2p^n$, 其中 p 是奇质数, n 是正整数

```
vector<int> findPrimitiveRoot(int N) {
  if (N <= 4) return vector<int>(1, max(1, N - 1));
  static int factor[100];
  int phi = N, totF = 0;
  { // check no solution and calculate phi
    int M = N, k = 0;
    if (~M & 1) M >>= 1, phi >>= 1;
    if (~M & 1) return vector<int>(0);
    for (int d = 3; d * d <= M; ++d) if (M % d == 0) {
      if (++k > 1) return vector<int>(0);
      for (phi -= phi / d; M % d == 0; M /= d);
    } if (M > 1) {
      if (++k > 1) return vector<int>(0); phi -= phi / M;
    }
  } { // factorize phi
```

```cpp
    int M = phi;
    for (int d = 2; d * d <= M; ++d) if (M % d == 0) {
      for ( ; M % d == 0; M /= d); factor[++totF] = d;
    } if (M > 1) factor[++totF] = M;
  } vector<int> ans;
  for (int g = 2; g <= N; ++g) if (Gcd(g, N) == 1) {
    bool good = true;
    for (int i = 1; i <= totF && good; ++i)
      if (powMod(g, phi / factor[i], N) == 1) good = false;
    if (!good) continue;
    for (int i = 1, gp = g; i <= phi; ++i, gp = (LL)gp * g % N)
      if (Gcd(i, phi) == 1) ans.push_back(gp);
    break;
  } sort(ans.begin(), ans.end());
  return ans;
}
```

5.11 离散对数

$A^x \equiv B \pmod{()}C$, 对非质数 C 也适用.

```cpp
int modLog(int A, int B, int C) {
  static pii baby[MAX_SQRT_C + 11];
  int d = 0; LL k = 1, D = 1; B %= C;
  for (int i = 0; i < 100; ++i, k = k * A % C) // [0,logC]
    if (k == B) return i;
  for (int g; ; ++d) {
    g = gcd(A, C); if (g == 1) break;
    if (B % g != 0) return -1;
    B /= g; C /= g; D = (A / g * D) % C;
  } int m = (int) ceil(sqrt((double) C)); k = 1;
  for (int i = 0; i <= m; ++i, k = k * A % C) baby[i] = pii(k, i);
  sort(baby, baby + m + 1); // [0, m]
  int n = unique(baby, baby + m + 1, equalFirst) - baby, am = powMod(A, m, C);
  for (int i = 0; i <= m; ++i) {
    LL e, x, y; exgcd(D, C, x, y, e); e = x * B % C;
    if (e < 0) e += C;
    if (e >= 0) {
      int k = lower_bound(baby, baby + n, pii(e, -1)) - baby;
      if (baby[k].first == e) return i * m + baby[k].second + d;
    } D = D * am % C;
  } return -1;
}
```

5.12 平方剩余

- Legrendre Symbol: 对奇质数 p, $(\frac{a}{p}) = \begin{cases} 1 & 是平方剩余 \\ -1 & 是非平方剩余 = a^{\frac{p-1}{2}} \bmod p \\ 0 & a \equiv 0 \pmod p \end{cases}$
- 若 p 是奇质数, $(\frac{-1}{p}) = 1$ 当且仅当 $p \equiv 1 \pmod 4$
- 若 p 是奇质数, $(\frac{2}{p}) = 1$ 当且仅当 $p \equiv \pm 1 \pmod 8$
- 若 p, q 是奇素数且互质, $(\frac{p}{q})(\frac{q}{p}) = (-1)^{\frac{p-1}{2} \times \frac{q-1}{2}}$
- Jacobi Symbol: 对奇数 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$, $(\frac{a}{n}) = (\frac{a}{p_1})^{\alpha_1} (\frac{a}{p_2})^{\alpha_2} \cdots (\frac{a}{p_k})^{\alpha_k}$
- Jacobi Symbol 为 -1 则一定不是平方剩余, 所有平方剩余的 Jacobi Symbol 都是 1, 但 1 不一定是平方剩余

$ax^2 + bx + c \equiv 0 \pmod p$, 其中 $a \neq 0 \pmod p$, 且 p 是质数

```cpp
inline int normalize(LL a, int P) { a %= P; return a < 0 ? a + P : a; }
vector<int> QuadraticResidue(LL a, LL b, LL c, int P) {
  int h, t; LL r1, r2, delta, pb = 0;
  a = normalize(a, P); b = normalize(b, P); c = normalize(c, P);
  if (P == 2) { vector<int> res;
    if (c % P == 0) res.push_back(0);
    if ((a + b + c) % P == 0) res.push_back(1);
    return res;
  } delta = b * rev(a + a, P) % P;
  a = normalize(-c * rev(a, P) + delta * delta, P);
  if (powMod(a, P / 2, P) + 1 == P) return vector<int>(0);
  for (t = 0, h = P / 2; h % 2 == 0; ++t, h /= 2);
  r1 = powMod(a, h / 2, P);
  if (t > 0) { do b = random() % (P - 2) + 2;
    while (powMod(b, P / 2, P) + 1 != P); }
  for (int i = 1; i <= t; ++i) {
    LL d = r1 * r1 % P * a % P;
    for (int j = 1; j <= t - i; ++j) d = d * d % P;
    if (d + 1 == P) r1 = r1 * pb % P; pb = pb * pb % P;
  } r1 = a * r1 % P; r2 = P - r1;
  r1 = normalize(r1 - delta, P); r2 = normalize(r2 - delta, P);
  if (r1 > r2) swap(r1, r2); vector<int> res(1, r1);
  if (r1 != r2) res.push_back(r2);
  return res;
}
```

5.13 N 次剩余

- 若 p 为奇质数, a 为 p 的 n 次剩余的充要条件是 $a^{\frac{p-1}{(a,p-1)}} \equiv 1 \pmod p$.

$x^N \equiv a \pmod p$, 其中 p 是质数

```cpp
vector<int> solve(int p, int N, int a) {
  if ((a %= p) == 0) return vector<int>(1, 0);
  int g = findPrimitiveRoot(p), m = modLog(g, a, p); // g ^ m = a (mod p)
  if (m == -1) return vector<int>(0);
  LL B = p - 1, x, y, d; exgcd(N, B, x, y, d);
  if (m % d != 0) return vector<int>(0);
  vector<int> ret; x = (x * (m / d) % B + B) % B; // g ^ B mod p = g ^ (p - 1) mod p = 1
  for (int i = 0, delta = B / d; i < d; ++i) {
    x = (x + delta) % B; ret.push_back((int)powMod(g, x, p));
  } sort(ret.begin(), ret.end());
  ret.resize(unique(ret.begin(), ret.end()) - ret.begin());
  return ret;
}
```

5.14 Romberg 积分

```
template <class T> double Romberg(const T&f, double a, double b, double eps =
    1e-8) {
  vector<double> t; double h = b - a, last, now; int k = 1, i = 1;
  t.push_back(h * (f(a) + f(b)) / 2); // 梯形
  do {
    last = t.back(); now = 0; double x = a + h / 2;
    for (int j = 0; j < k; ++j, x += h) now += f(x);
    now = (t[0] + h * now) / 2; double k1 = 4.0 / 3.0, k2 = 1.0 / 3.0;
    for (int j = 0; j < i; ++j, k1 = k2 + 1) {
      double tmp = k1 * now - k2 * t[j];
      t[j] = now; now = tmp; k2 /= 4 * k1 - k2; // 防止溢出
    } t.push_back(now); k *= 2; h /= 2; ++i;
  } while (fabs(last - now) > eps);
  return t.back();
}
```

5.15 公式

5.15.1 级数与三角

- $\sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
- $\sum_{k=1}^{n} k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$
- 错排: $D_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^n}{n!}) = (n-1)(D_{n-2} - D_{n-1})$
- $\tan\alpha \pm \tan\beta = \frac{\sin(\alpha\pm\beta)}{\cos\alpha\cos\beta}$
- $\cos n\alpha = \binom{n}{0}\cos^n\alpha - \binom{n}{2}\cos^{n-2}\alpha\sin^2\alpha + \binom{n}{4}\cos^{n-4}\alpha\sin^4\alpha\cdots$
- $\sin n\alpha = \binom{n}{1}\cos^{n-1}\alpha\sin\alpha - \binom{n}{3}\cos^{n-3}\alpha\sin^3\alpha + \binom{n}{5}\cos^{n-5}\alpha\sin^5\alpha\cdots$
- $\sum_{n=1}^{N}\cos nx = \frac{\sin(N+\frac{1}{2})x - \sin\frac{x}{2}}{2\sin\frac{x}{2}}$
- $\sum_{n=1}^{N}\sin nx = \frac{-\cos(N+\frac{1}{2})x + \cos\frac{x}{2}}{2\sin\frac{x}{2}}$
- $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}\cdots$ for $x \in (-\infty, +\infty)$
- $\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!}\cdots$ for $x \in (-\infty, +\infty)$
- $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!}\cdots$ for $x \in (-\infty, +\infty)$
- $\arcsin x = x + \sum_{n=1}^{\infty}\frac{(2n-1)!!}{(2n)!!}\frac{x^{2n+1}}{2n+1}$ for $x \in [-1, 1]$
- $\arccos x = \frac{\pi}{2} - \sum_{n=1}^{\infty}\frac{(2n-1)!!}{(2n)!!}\frac{x^{2n+1}}{2n+1}$ for $x \in [-1, 1]$
- $\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5}\cdots$ for $x \in [-1, 1]$
- $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3}\cdots$ for $x \in (-1, 1]$

- $\int_0^{\frac{\pi}{2}} \sin^n x\mathrm{d}x = \begin{cases} \frac{(n-1)!!}{n!!}\times\frac{\pi}{2} & n\text{是偶数} \\ \frac{(n-1)!!}{n!!} & n\text{是奇数} \end{cases}$
- $\int_0^{+\infty}\frac{\sin x}{x}\mathrm{d}x = \frac{\pi}{2}$
- $\int_0^{+\infty} e^{-x^2}\mathrm{d}x = \frac{\sqrt{\pi}}{2}$
- 傅里叶级数: 设周期为 $2T$. 函数分段连续. 在不连续点的值为左右极限的平均数.
  - $a_n = \frac{1}{T}\int_{-T}^{T} f(x)\cos\frac{n\pi}{T}x\mathrm{d}x$
  - $b_n = \frac{1}{T}\int_{-T}^{T} f(x)\sin\frac{n\pi}{T}x\mathrm{d}x$
  - $f(x) = \frac{a_0}{2} + \sum_{n=1}^{+\infty}(a_n\cos\frac{n\pi}{T}x + b_n\sin\frac{n\pi}{T}x)$
- Beta 函数: $B(p,q) = \int_0^1 x^{p-1}(1-x)^{q-1}\mathrm{d}x$
  - 定义域 $(0,+\infty)\times(0,+\infty)$, 在定义域上连续
  - $B(p,q) = B(q,p) = \frac{q-1}{p+q-1}B(p,q-1) = 2\int_0^{\frac{\pi}{2}}\cos^{2p-1}\phi\sin^{2p-1}\phi\mathrm{d}\phi = \int_0^{+\infty}\frac{t^{q-1}}{(1+t)^{p+q}}\mathrm{d}t = \int_0^1\frac{t^{p-1}+t^{q-1}}{(1+t)^(p+q)}$
  - $B(\frac{1}{2},\frac{1}{2}) = \pi$
- Gamma 函数: $\Gamma = \int_0^{+\infty} x^{s-1}e^{-x}\mathrm{d}x$
  - 定义域 $(0,+\infty)$, 在定义域上连续
  - $\Gamma(1) = 1$, $\Gamma(\frac{1}{2}) = \sqrt{\pi}$
  - $\Gamma(s) = (s-1)\Gamma(s-1)$
  - $B(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}$
  - $\Gamma(s)\Gamma(1-s) = \frac{\pi}{\sin\pi s}$ for $s > 0$
  - $\Gamma(s)\Gamma(s+\frac{1}{2}) = 2\sqrt{\pi}\frac{\Gamma(s)}{2^{2s-1}}$ for $0 < s < 1$
-

	$y = f(x)$	$x = x(t), y = y(t), t \in [T_1, T_2]$	$r = r(\theta), \theta \in [\alpha, \beta]$
平面图形面积	$\int_a^b f(x)\mathrm{d}x$	$\int_{T_1}^{T_2}\lvert y(t)x'(t)\rvert\mathrm{d}t$	$\frac{1}{2}\int_\alpha^\beta r^2(\theta)\mathrm{d}\theta$
曲线弧长	$\int_a^b\sqrt{1+f'^2(x)}\mathrm{d}x$	$\int_{T_1}^{T_2}\sqrt{x'^2(t)+y'^2(t)}\mathrm{d}t$	$\int_\alpha^\beta\sqrt{r^2(\theta)+r'^2(\theta)}\mathrm{d}\theta$
旋转体体积	$\pi\int_a^b f^2(x)\mathrm{d}x$	$\pi\int_{T_1}^{T_2}\lvert x'(t)\rvert y^2(t)\mathrm{d}t$	$\frac{2}{3}\pi\int_\alpha^\beta r^3(\theta)\sin\theta\mathrm{d}\theta$
旋转曲面面积	$2\pi\int_a^b\lvert f(x)\rvert\sqrt{1+f'^2(x)}\mathrm{d}x$	$2\pi\int_{T_1}^{T_2}\lvert y(t)\rvert\sqrt{x'^2(t)+y'^2(t)}\mathrm{d}t$	$2\pi\int_\alpha^\beta r(\theta)\sin\theta\sqrt{r^2(\theta)+r'^2(\theta}$

5.15.2 三次方程求根公式

对一元三次方程 $x^3+px+q=0$, 令

$$A=\sqrt[3]{-\frac{q}{2}+\sqrt{(\frac{q}{2})^2+(\frac{p}{3})^3}}$$
$$B=\sqrt[3]{-\frac{q}{2}-\sqrt{(\frac{q}{2})^2+(\frac{p}{3})^3}}$$
$$\omega=\frac{(-1+\mathrm{i}\sqrt{3})}{2}$$

则 $x_j=A\omega^j+B\omega^{2j}$ (j = 0, 1, 2).
当求解 $ax^3+bx^2+cx+d=0$ 时, 令 $x=y-\frac{b}{3a}$, 再求解 y, 即转化为 $y^3+py+q=0$ 的形式. 其中,

$$p=\frac{b^2-3ac}{3a^2}$$
$$q=\frac{2b^3-9abc+27a^2d}{27a^3}$$

卡尔丹判别法: 令 $\Delta=(\frac{q}{2})^2+(\frac{p}{3})^3$. 当 $\Delta>0$ 时, 有一个实根和一对个共轭虚根; 当 $\Delta=0$ 时, 有三个实根, 其中两个相等; 当 $\Delta<0$ 时, 有三个不相等的实根.

5.15.3 椭圆

- 椭圆 $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$, 其中离心率 $e=\frac{c}{a}, c=\sqrt{a^2-b^2}$; 焦点参数 $p=\frac{b^2}{a}$
- 椭圆上 (x,y) 点处的曲率半径为 $R=a^2b^2(\frac{x^2}{a^4}+\frac{y^2}{b^4})^{\frac{3}{2}}=\frac{(r_1r_2)^{\frac{3}{2}}}{ab}$, 其中 r_1 和 r_2 分别为 (x,y) 与两焦点 F_1 和 F_2 的距离.
- 椭圆的周长 $L=4a\int_0^{\frac{\pi}{2}}\sqrt{1-e^2\sin^2 t}\mathrm{d}t=4aE(e,\frac{\pi}{2})$, 其中

$$E(e,\frac{\pi}{2})=\frac{\pi}{2}[1-(\frac{1}{2})^2e^2-(\frac{1\times 3}{2\times 4})^2\frac{e^4}{3}-(\frac{1\times 3\times 5}{2\times 4\times 6})^2\frac{e^6}{5}-\cdots$$

- 设椭圆上点 $M(x,y), N(x,-y), x,y>0, A(a,0)$, 原点 $O(0,0)$, 扇形 OAM 的面积 $S_{OAM}=\frac{1}{2}ab\arccos\frac{x}{a}$, 弓形 MAN 的面积 $S_{MAN}=ab\arccos\frac{x}{a}-xy$.
- 设 θ 为 (x,y) 点关于椭圆中心的极角, r 为 (x,y) 到椭圆中心的距离, 椭圆极坐标方程:

$$x=r\cos\theta, y=r\sin\theta, r^2=\frac{b^2a^2}{b^2\cos^2\theta+a^2\sin^2\theta}$$

5.15.4 抛物线

- 标准方程 $y^2=2px$, 曲率半径 $R=\frac{(p+2x)^{\frac{3}{2}}}{\sqrt{p}}$
- 弧长: 设 $M(x,y)$ 是抛物线上一点, 则 $L_{OM}=\frac{p}{2}[\sqrt{\frac{2x}{p}(1+\frac{2x}{p})}+\ln(\sqrt{\frac{2x}{p}}+\sqrt{1+\frac{2x}{p}})]$
- 弓形面积: 设 M,D 是抛物线上两点, 且分居一, 四象限. 做一条平行于 MD 且与抛物线相切的直线 L. 若 M 到 L 的距离为 h. 则有 $S_{MOD}=\frac{2}{3}MD\cdot h$.

5.15.5 重心

- 半径 r, 圆心角为 θ 的扇形的重心与圆心的距离为 $\frac{4r\sin\frac{\theta}{2}}{3\theta}$
- 半径 r, 圆心角为 θ 的圆弧的重心与圆心的距离为 $\frac{4r\sin^3\frac{\theta}{2}}{3(\theta-\sin\theta)}$
- 椭圆上半部分的重心与圆心的距离为 $\frac{4b}{3\pi}$
- 抛物线中弓形 MOD 的重心满足 $CQ=\frac{2}{5}PQ$, P 是直线 L 与抛物线的切点, Q 在 MD 上且 PQ 平行 x 轴, C 是重心

5.15.6 向量恒等式

- $\vec{a}\cdot(\vec{b}\times\vec{c})=\vec{b}\cdot(\vec{c}\times\vec{a})=\vec{c}\cdot(\vec{a}\times\vec{b})$
- $\vec{a}\times(\vec{b}\times\vec{c})=(\vec{c}\times\vec{b})\times\vec{a}=\vec{b}(\vec{a}\cdot\vec{c})-\vec{c}(\vec{a}\cdot\vec{b})$

5.15.7 常用几何公式

- 三角形的五心
  - 重心 $\vec{G}=\frac{\vec{A}+\vec{B}+\vec{C}}{3}$
  - 内心 $\vec{I}=\frac{a\vec{A}+b\vec{B}+c\vec{C}}{a+b+c}$, $R=\frac{2S}{a+b+c}$
  - 外心 $\vec{O}=\frac{\vec{A}+\vec{B}+\frac{\vec{AC}\cdot\vec{BC}}{\vec{AB}\times\vec{BC}}\vec{AB}^T}{2}$, $R=\frac{abc}{4S}$
  - 垂心 $\vec{H}=3\vec{G}-2\vec{O}=\vec{C}+\frac{\vec{BC}\cdot\vec{AC}}{\vec{BC}\times\vec{AC}}\vec{AB}^T$
  - 旁心 (三个) $\frac{-a\vec{A}+b\vec{B}+c\vec{C}}{-a+b+c}$
- 四边形: 设 D_1, D_2 为对角线, M 为对角线中点连线, A 为对角线夹角
  - $a^2+b^2+c^2+d^2=D_1^2+D_2^2+4M^2$
  - $S=\frac{1}{2}D_1D_2\sin A$
  - $ac+bd=D_1D_2$ (内接四边形适用)
  - Bretschneider 公式: $S=\sqrt{(p-a)(p-b)(p-c)(p-d)-abcd\cos^2(\frac{\theta}{2})}$, 其中 θ 为对角和

5.15.8 树的计数

- 有根数计数: 令 $S_{n,j}=\sum\limits_{1\le i\le n/j}a_{n+1-ij}=S_{n-j,j}+a_{n+1-j}$
  于是, $n+1$ 个结点的有根数的总数为 $a_{n+1}=\frac{\sum\limits_{1\le j\le n}j\cdot a_j\cdot S_{n,j}}{n}$
  附: $a_1=1, a_2=1, a_3=2, a_4=4, a_5=9, a_6=20, a_9=286, a_{11}=1842$
- 无根树计数: 当 n 是奇数时, 则有 $a_n-\sum\limits_{1\le i\le\frac{n}{2}}a_ia_{n-i}$ 种不同的无根树
  当 n 是偶数时, 则有 $a_n-\sum\limits_{1\le i\le\frac{n}{2}}a_ia_{n-i}+\frac{1}{2}a_{\frac{n}{2}}(a_{\frac{n}{2}}+1)$ 种不同的无根树
- Matrix-Tree 定理: 对任意图 G, 设 $\mathrm{mat}[i][i]=i$ 的度数, $\mathrm{mat}[i][j]=i$ 与 j 之间边数的相反数, 则 $\mathrm{mat}[i][j]$ 的任意余子式的行列式就是该图的生成树个数

5.16 小知识

- 勾股数: 设正整数 n 的质因数分解为 $n = \prod p_i^{a_i}$, 则 $x^2 + y^2 = n$ 有整数解的充要条件是 n 中不存在形如 $p_i \equiv 3 \pmod 4$ 且指数 a_i 为奇数的质因数 p_i

- 勾股数 2:

```
a[0] := 0;
s := 0;
for i := 1 to n - 2 do
  begin
    a[i] := a[i - 1] + 1;
    s := s + sqr(a[i]);
  end;
{======s + sqr(a[n-1]) + sqr(a[n]) = k^2=======}
a[n - 1] := a[n - 2];
repeat
  a[n - 1] := a[n - 1] + 1;
until odd(s + sqr(a[n - 1])) and (a[n - 1] > 2);
a[n] := (s + sqr(a[n - 1]) - 1) shr 1;
```

  知道 `s` 和 `a[n-1]` 后, 直接求了 `a[n]`. 神奇了点.

  其实, 有当 n 为奇数: $n^2 + \lfloor \frac{n^2-1}{2} \rfloor^2 = \lfloor \frac{n^2+1}{2} \rfloor^2$

  若:

  $a = k \cdot (s^2 - t^2)$

  $b = 2 \cdot k \cdot s \cdot t$

  $c = k \cdot (s^2 + t^2)$

  则 $c^2 = a^2 + b^2$.

- Stirling 公式: $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$

- Pick 定理: 简单多边形, 不自交, 顶点如果全是整点. 则: 严格在多边形内部的整点数$+\frac{1}{2}$在边上的整点数$-1=$面积

- Mersenne 素数: p 是素数且 $2^p - 1$ 的数是素数. (10000 以内的 p 有: 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941)

- Fermat 分解算法: 从 $t = \sqrt{n}$ 开始, 依次检查 $t^2 - n, (t+1)^2 - n, (t+2)^2 - n, \ldots$, 直到出现一个平方数 y, 由于 $t^2 - y^2 = n$, 因此分解得 $n = (t-y)(t+y)$. 显然, 当两个因数很接近时这个方法能很快找到结果, 但如果遇到一个素数, 则需要检查 $\frac{n+1}{2} - \sqrt{n}$ 个整数

- 牛顿迭代: $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

- 球与盒子的动人故事: (n 个球, m 个盒子, S 为第二类斯特林数)
  1. 球同, 盒同, 无空: dp
  2. 球同, 盒同, 可空: dp
  3. 球同, 盒不同, 无空: $\binom{n-1}{m-1}$
  4. 球同, 盒不同, 可空: $\binom{n+m-1}{n-1}$
  5. 球不同, 盒同, 无空: $S(n, m)$
  6. 球不同, 盒同, 可空: $\sum_{k=1}^{m} S(n, k)$
  7. 球不同, 盒不同, 无空: $m!S(n, m)$
  8. 球不同, 盒不同, 可空: m^n

- 组合数:
  - Lucas 定理: 对质数 p, 设 $n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$, $m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$, 则 $\binom{n}{m} \equiv \prod i = 0^k \binom{n_i}{m_i} \pmod p$
  - 组合数判断奇偶性: 若 $(n\&m) = m$, 则 $\binom{n}{m}$ 为奇数, 否则为偶数

- 格雷码 $G(x) = x \otimes (x >> 1)$

- Bell 数: B_n 代表将 n 个元素划分成若干个非空集合的方案数
  - $B_0 = B_1 = 1$, $B_n = \sum\limits_{k=0}^{n-1} \binom{n-1}{k} B_k$
  - $B_n = \sum\limits_{k=0}^{n} \left\{ {n \atop k} \right\}$
  - Bell 三角形: $a_{1,1} = 1$, $a_{n,1} = a_{n-1,n-1}$, $a_{n,m} = a_{n,m-1} + a_{n-1,m-1}$, $B_n = a_{n,1}$
  - 对质数 p, $B_{n+p} \equiv B_n + B_{n+1} \pmod p$
  - 对质数 p, $B_{n+p^m} \equiv mB_n + B_{n+1} \pmod p$
  - 对质数 p, 模的周期一定是 $\frac{p^p-1}{p-1}$ 的约数, $p \leq 101$ 时就是这个值
  - 从 B_0 开始, 前几项是 $1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975 \cdots$

6 其他

6.1 Extended LIS

```
int G[MAXN][MAXN];
void insertYoung(int v) {
  for (int x = 1, y = INT_MAX; ; ++x) {
    Down(y, *G[x]); while (y > 0 && G[x][y] >= v) --y;
    if (++y > *G[x]) { ++*G[x]; G[x][y] = v; break; }
    else swap(G[x][y], v);
  }
}
int solve(int N, int seq[]) {
  Rep(i, 1, N) *G[i] = 0;
  Rep(i, 1, N) insertYoung(seq[i]);
  printf("%d\n", *G[1] + *G[2]);
  return 0;
}
```

6.2 生成 nCk

```
void nCk(int n, int k) {
  for (int comb = (1 << k) - 1; comb < (1 << n); ) {
    int x = comb & -comb, y = comb + x;
    comb = (((comb & ~y) / x) >> 1) | y;
  }
}
```

6.3 nextPermutation

```
boolean nextPermutation(int[] is) {
  int n = is.length;
  for (int i = n - 1; i > 0; i--) {
    if (is[i - 1] < is[i]) {
      int j = n; while (is[i - 1] >= is[--j]);
      swap(is, i - 1, j); // swap is[i - 1], is[j]
      rev(is, i, n); // reverse is[i, n)
      return true;
    }
  } rev(is, 0, n);
  return false;
}
```

6.4 Josephus 数与逆 Josephus 数

```
int josephus(int n, int m, int k) { int x = -1;
  for (int i = n - k + 1; i <= n; i++) x = (x + m) % i; return x;
}
int invJosephus(int n, int m, int x) {
  for (int i = n; ; i--) { if (x == i) return n - i; x = (x - m % i + i) % i;
      }
}
```

6.5 表达式求值

```
inline int getLevel(char ch) {
  switch (ch) { case '+': case '-': return 0; case '*': return 1; } return
      -1;
}
int evaluate(char *&p, int level) {
  int res;
  if (level == 2) {
    if (*p == '(') ++p, res = evaluate(p, 0);
    else res = isdigit(*p) ? *p - '0' : value[*p - 'a'];
    ++p; return res;
  } res = evaluate(p, level + 1);
  for (int next; *p && getLevel(*p) == level; ) {
    char op = *p++; next = evaluate(p, level + 1);
    switch (op) {
      case '+': res += next; break;
      case '-': res -= next; break;
      case '*': res *= next; break;
    }
  } return res;
}
int makeEvaluation(char *str) { char *p = str; return evaluate(p, 0); }
```

6.6 直线下的整点个数

求 $\sum_{i=0}^{n-1}\lfloor\frac{a+bi}{m}\rfloor$

```
LL count(LL n, LL a, LL b, LL 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);
}
```

6.7 Java 多项式

```
class Polynomial {
  final static Polynomial ZERO = new Polynomial(new int[] { 0 });
  final static Polynomial ONE = new Polynomial(new int[] { 1 });
  final static Polynomial X = new Polynomial(new int[] { 0, 1 });
  int[] coef;
  static Polynomial valueOf(int val) { return new Polynomial(new int[] { val
      }); }
  Polynomial(int[] coef) { this.coef = Arrays.copyOf(coef, coef.length); }
  Polynomial add(Polynomial o, int mod); // omitted
  Polynomial subtract(Polynomial o, int mod); // omitted
  Polynomial multiply(Polynomial o, int mod); // omitted
  Polynomial scale(int o, int mod); // omitted
  public String toString() {
    int n = coef.length; String ret = "";
    for (int i = n - 1; i > 0; --i) if (coef[i] != 0)
      ret += coef[i] + "x^" + i + "+";
    return ret + coef[0];
  }
  static Polynomial lagrangeInterpolation(int[] x, int[] y, int mod) {
    int n = x.length; Polynomial ret = Polynomial.ZERO;
    for (int i = 0; i < n; ++i) {
      Polynomial poly = Polynomial.valueOf(y[i]);
      for (int j = 0; j < n; ++j) if (i != j) {
        poly = poly.multiply(
          Polynomial.X.subtract(Polynomial.valueOf(x[j]), mod), mod);
        poly = poly.scale(powMod(x[i] - x[j] + mod, mod - 2, mod), mod);
      } ret = ret.add(poly, mod);
    } return ret;
  }
}
```

6.8 long long 乘法取模

```
LL multiplyMod(LL a, LL b, LL P) { // 需要保证 a 和 b 非负
  LL t = (a * b - LL((long double)a / P * b + 1e-3) * P) % P;
  return t < 0 : t + P : t;
}
```

6.9 重复覆盖

```
struct node { int x, y; node *l, *r, *u, *d; } base[MAX * MAX], *top, *head;
typedef node *link;
int row, col, nGE, ans, stamp, cntc[MAX], vis[MAX];
void removeExact(link c) { c->l->r = c->r; c->r->l = c->l;
  for (link i = c->d; i != c; i = i->d)
    for (link j = i->r; j != i; j = j->r) j->d->u = j->u, j->u->d = j->d, --
        cntc[j->y];
```

```
}
void resumeExact(link c) {
    for (link i = c->u; i != c; i = i->u)
        for (link j = i->l; j != i; j = j->l) j->d->u = j, j->u->d = j, ++cntc[j->y];
    c->l->r = c; c->r->l = c;
}
void removeRepeat(link c) { for (link i = c->d; i != c; i = i->d) i->l->r = i->r, i->r->l = i->l; }
void resumeRepeat(link c) { for (link i = c->u; i != c; i = i->u) i->l->r = i; i->r->l = i; }
int calcH() { int y, res = 0; ++stamp;
    for (link c = head->r; (y = c->y) <= row && c != head; c = c->r) if (vis[y] != stamp) {
        vis[y] = stamp; ++res; for (link i = c->d; i != c; i = i->d)
            for (link j = i->r; j != i; j = j->r) vis[j->y] = stamp;
    } return res;
}
void DFS(int dep) { if (dep + calcH() >= ans) return;
    if (head->r->y > nGE || head->r == head) { if (ans > dep) ans = dep; return; }
    link c = NULL;
    for (link i = head->r; i->y <= nGE && i != head; i = i->r)
        if (!c || cntc[i->y] < cntc[c->y]) c = i;
    for (link i = c->d; i != c; i = i->d) {
        removeRepeat(i);
        for (link j = i->r; j != i; j = j->r) if (j->y <= nGE) removeRepeat(j);
        for (link j = i->r; j != i; j = j->r) if (j->y > nGE) removeExact(base + j->y);
        DFS(dep + 1);
        for (link j = i->l; j != i; j = j->l) if (j->y > nGE) resumeExact(base + j->y);
        for (link j = i->l; j != i; j = j->l) if (j->y <= nGE) resumeRepeat(j);
        resumeRepeat(i);
    }
}
```

6.10 星期几判定

```
int getDay(int y, int m, int d) {
    if (m <= 2) m += 12, y--;
    if (y < 1752 || (y == 1752 && m < 9) || (y == 1752 && m == 9 && d < 3))
        return (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 + 5) % 7 + 1;
    return (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 - y / 100 + y / 400) % 7 + 1;
}
```

6.11 LCSequence Fast

```
ULL *a, *b, *s, c, d;
for (i = 0, a = appear[(int)B[k]], b = row[max(k - 1, 0)], s = X; i < bitSetLen; ++i)
    *s++ = *a++ | *b++;                          // X = row[i - 1] or appear[ B[i] ]
for (i = 0, a = dp, c = d = 0; i < bitSetLen; ++a, c = d, ++i)
    d = *a >> 63, *a = ~((*a << 1) + c);         // row[i] = -((row[i] << 1) + 1)
for (i = 0, a = dp, b = X, c = 0; i < bitSetLen; ++a, ++b, ++i)
    d = *b + c, c = (*a >= -d), *a += d;         // row[i] += X
for (i = 0, a = dp, b = X; i < bitSetLen; ++a, ++b, ++i)
    *a = (*a ^ *b) & *b;                         // row[i] = X and (row[i] xor X)
```

7 Templates

7.1 vim 配置

在.bashrc 中加入 export CXXFLAGS="-Wall -Wconversion -Wextra -g3"

```
set nu ru nobk cindent si
set mouse=a sw=4 sts=4 ts=4
set hlsearch incsearch
set whichwrap=b,s,<,>,[,]
syntax on

nmap <C-A> ggVG
vmap <C-C> "+y

autocmd␣BufNewFile␣*.cpp␣0r␣~/Templates/cpp.cpp
map<F9>␣:!g++␣%␣-o␣%<␣-Wall␣-Wconversion␣-Wextra␣-g3␣<CR>
map<F5>␣:!./%<␣<CR>
map<F8>␣:!./%<␣<␣%<.in␣<CR>

map<F3>␣:vnew␣%<.in␣<CR>
map<F4>␣:!(gedit␣%␣&)<CR>
```

7.2 C++

```
#pragma comment(linker, "/STACK:10240000")
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <algorithm>
#define Rep(i, a, b) for(int i = (a); i <= (b); ++i)
#define Foru(i, a, b) for(int i = (a); i < (b); ++i)
using namespace std;
typedef long long LL;
typedef pair<int, int> pii;
namespace BufferedReader {
    char buff[MAX_BUFFER + 5], *ptr = buff, c; bool flag;
    bool nextChar(char &c) {
        if ( (c = *ptr++) == 0 ) {
            int tmp = fread(buff, 1, MAX_BUFFER, stdin);
            buff[tmp] = 0; if (tmp == 0) return false;
            ptr = buff; c = *ptr++;
        } return true;
    }
    bool nextUnsignedInt(unsigned int &x) {
        for (;;){if (!nextChar(c)) return false; if ('0'<=c && c<='9') break;}
        for (x=c-'0'; nextChar(c); x = x * 10 + c - '0') if (c < '0' || c > '9') break;
        return true;
    }
    bool nextInt(int &x) {
        for (;;) { if (!nextChar(c)) return false; if (c=='-' || ('0'<=c && c<='9')) break; }
        for ((c=='-') ? (x=0,flag=true) : (x=c-'0',flag=false); nextChar(c); x=x*10+c-'0')
```

```
      if (c<'0' || c>'9') break;
    if (flag) x=-x; return true;
  }
};
#endif
```

7.3 Java

```
import java.io.*;
import java.util.*;
import java.math.*;

public class Main {
  public void solve() {}
  public void run() {
    tokenizer = null; out = new PrintWriter(System.out);
    in = new BufferedReader(new InputStreamReader(System.in));
    solve();
    out.close();
  }
  public static void main(String[] args) {
    new Main().run();
  }
  public StringTokenizer tokenizer;
  public BufferedReader in;
  public PrintWriter out;
  public String next() {
    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
      try { tokenizer = new StringTokenizer(in.readLine()); }
      catch (IOException e) { throw new RuntimeException(e); }
    } return tokenizer.nextToken();
  }
}
```