## Angle CircleIntersection CircleTangents CircleLine CirclePolygonIntersection circumcircle MinimumEnclosingCircle InsidePolygon PolygonArea

**Description:** Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h" b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
 P v = b - a;
 return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
```

## Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage: vector<Angle>  $v = \{w[0], w[0].t360() ...\}; // sorted$ int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; } // sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i 0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
   assert(x || y);
    return v < 0 || (v == 0 && x < 0);
  Angle t90() const { return \{-v, x, t + (half() \&\& x >= 0)\}; \}
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
bool operator<(Angle a, Angle b) {</pre>
  // add a.dist2() and b.dist2() to also compare distances
  return make tuple(a.t, a.half(), a.v * (11)b.x) <</pre>
         make_tuple(b.t, b.half(), a.x * (11)b.y);
// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair < Angle, Angle > segment Angles (Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
Angle operator+(Angle a, Angle b) { // point \ a + vector \ b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;</pre>
  return r.t180() < a ? r.t360() : r;
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
 return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
```

## 7.2 Circles

"Point.h"

#### CircleIntersection.h.

Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

84d6d3, 11 lines typedef Point < double > P; bool circleInter(P a, P b, double r1, double r2, pair < P, P >\* out) { if (a == b) { assert(r1 != r2); return false; } P vec = b - a;double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2, p = (d2 + r1\*r1 - r2\*r2)/(d2\*2), h2 = r1\*r1 - p\*p\*d2;

```
if (sum*sum < d2 || dif*dif > d2) return false;
P mid = a + \text{vec*p}, per = \text{vec.perp}() * \text{sqrt}(\text{fmax}(0, h2) / d2);
*out = {mid + per, mid - per};
return true:
```

#### CircleTangents.h

Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h" b0153d, 13 lines template<class P> vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) { P d = c2 - c1;double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr \* dr; if (d2 == 0 || h2 < 0) return {};</pre> vector<pair<P, P>> out; for (double sign : {-1, 1}) { P v = (d \* dr + d.perp() \* sqrt(h2) \* sign) / d2;out.push\_back( $\{c1 + v * r1, c2 + v * r2\}$ ); **if** (h2 == 0) out.pop back(); return out;

### CircleLine.h

Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point < double >.

```
"Point.h"
                                                       e0cfba, 9 lines
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
 P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
 double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
 if (h2 < 0) return {};
 if (h2 == 0) return {p};
 P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
```

#### CirclePolygonIntersection.h

Description: Returns the area of the intersection of a circle with a ccw polygon.

## Time: $\mathcal{O}(n)$

return sum;

```
"../../content/geometry/Point.h"
                                                                 a1ee63, 19 lines
typedef Point<double> P;
```

```
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&] (P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;</pre>
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, g) * r2;</pre>
    Pu = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,g) * r2;
  auto sum = 0.0;
  rep(i, 0, sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
```

## circumcircle.h

#### Description:

"Point.h"

The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



```
typedef Point<double> P;
double ccRadius (const P& A, const P& B, const P& C) {
  return (B-A).dist() * (C-B).dist() * (A-C).dist() /
      abs((B-A).cross(C-A))/2;
P ccCenter (const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
```

#### MinimumEnclosingCircle.h

**Description:** Computes the minimum circle that encloses a set of points. Time: expected  $\mathcal{O}(n)$ 

```
"circumcircle.h"
                                                     09dd0a, 17 lines
pair<P, double> mec(vector<P> ps) {
 shuffle(all(ps), mt19937(time(0)));
 P \circ = ps[0];
 double r = 0, EPS = 1 + 1e-8;
 rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
       o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
 return {o, r};
```

# 7.3 Polygons

bool in = inPolygon(v,  $P{3, 3}$ , false);

#### InsidePolygon.h

Time:  $\mathcal{O}(n)$ 

**Description:** Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow. Usage: vector $\langle P \rangle$  v =  $\{P\{4,4\}, P\{1,2\}, P\{2,1\}\};$ 

```
"Point.h", "OnSegment.h", "SegmentDistance.h"
                                                       2bf504, 11 lines
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
 int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) \le eps) return ! strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
 return cnt;
```

## PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T! "Point.h"