Geometry

Created	@2025년 5월 18일 오후 7:40
Tags	BasicOps ConvexHull Pick'sTheorem PointInPolygon PolygonCut RotatingCalipers SortingPoints

Basic Operations

```
const ld eps = 1e-9;
inline ll diff(ld lhs, ld rhs) {
    if (lhs-eps<rhs && rhs<lhs+eps) return 0;
    return lhs<rhs ? -1 : 1;
}
struct Point {
    ld x, y;
    bool operator==(const Point& o) const {
        return diff(x,o.x)==0 && diff(y, o.y)==0;
    }
    Point operator+(const Point& o) const {
        return Point{ x+o.x, y+o.y };
    }
    Point operator-(const Point& o) const {
        return Point{ x-o.x, y-o.y };
    }
    Point operator*(ld t) const {
        return Point{ x*t, y*t };
```

```
    }
    friend istream& operator>>(istream& is, Point& p) {
        return is >> p.x >> p.y;
    }
};
struct Line {
    Point pos, dir;
    Line()=default;
    Line(const Point& a, const Point& b): pos(a), dir(b - a) {}
    friend istream& operator>>(istream& is, Line& l) {
        Point p1, p2;
        is >> p1 >> p2;
        l.pos = p1;
        l.dir = Point{ p2.x-p1.x, p2.y-p1.y };
        return is;
    }
};
struct Circle {
    Point center;
    ld r;
};
inline ld inner(const Point& a, const Point& b) {
    return a.x*b.x+a.y*b.y;
}
inline ld outer(const Point& a, const Point& b) {
    return a.x*b.y-a.y*b.x;
}
// point-line positional relationship, (+) l, (-) r, (0) on line
inline ll ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point-line.pos), 0);
}
// a→b→c rotation orientation, (+) CCW, (-) CW, (0) collinear
inline ll ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b-a, c-a), 0);
}
inline bool is_between(ld check, ld a, ld b) {
    if (a<b) return a-eps<check && check<b+eps;
    return b-eps<check && check<a+eps;
```

```
}
// Euclidean distance
inline ld dist(const Point& a, const Point& b) {
    return sqrt(inner(a-b, a-b));
}
// Euclidean distance squared
inline ld dist2(const Point &a, const Point &b) {
    return inner(a-b, a-b);
}
// point-line distance, point-segment distance
inline ld dist(const Line& line, const Point& point, bool segment = false) {
    ld c1 = inner(point-line.pos, line.dir);
    if (segment && diff(c1, 0)<=0) return dist(line.pos, point);
    ld c2 = inner(line.dir, line.dir);
    if (segment && diff(c2, c1)<=0) return dist(line.pos+line.dir, point);
    return dist(line.pos+line.dir*(c1/c2), point);
}
// line intersection test, ret: intersection point
bool line_line(const Line& a, const Line& b, Point& ret) {
    ld mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    ld t2 = outer(a.dir, b.pos-a.pos)/mdet;
    ret = b.pos+b.dir*t2;
    return true;
}
// segment intersection test, ret: intersection point
// 0: no intersection, 1: single point, 2: multiple points
ll seg_seg(const Line& a, const Line& b, Point& ret) {
    ld mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) {
        if (ccw_line(a,b.pos)!=0) return 0;
        ld t0 = inner(b.pos-a.pos, a.dir)/inner(a.dir,a.dir);
        ld t1 = inner(b.pos+b.dir-a.pos,a.dir) / inner(a.dir,a.dir);
        if (t0>t1) swap(t0,t1);
        ld l=max((ld)0,t0),h=min((ld)1,t1);
        if (diff(h,l)<0) return 0;
        if (diff(h,l)>0) return 2;
        ret = a.pos+a.dir*l;
```

```
        return 1;
    }
    ld t1 = -outer(b.pos-a.pos, b.dir)/mdet;
    ld t2 = outer(a.dir, b.pos-a.pos)/mdet;
    if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return 0;
    ret = b.pos+b.dir*t2;
    return 1;
}
// line-point intersection test
inline bool line_point(const Line& line,const Point& point) {
    return ccw_line(line,point)==0;
}
// seg-point intersection test
inline bool seg_point(const Line& seg, const Point& point) {
    if (ccw_line(seg,point) != 0) return false;
    ld t = inner(point - seg.pos, seg.dir) / inner(seg.dir,seg.dir);
    return is_between(t,0,1);
}
// line-seg intersection test, ret: intersection point
// 0: no intersection, 1: single point, 2: multiple points
ll line_seg(const Line& line, const Line& seg, Point& ret) {
    ld mdet = outer(line.dir,seg.dir);
    if (diff(mdet,0)==0) {
        if (ccw_line(line,seg.pos)!=0) return 0;
        return 2;
    }
    ld t1 = -outer(line.pos-seg.pos,line.dir) / mdet;
    if (!is_between(t1,0,1)) return 0;
    ret = seg.pos + seg.dir * t1;
    return 1;
}
// circle-line intersection
vector<Point> circle_line(const Circle& circle, const Line& line) {
    vector<Point> result;
    ld a = 2*inner(line.dir, line.dir);
    ld b = 2*(line.dir.x*
        (line.pos.x-circle.center.x+line.dir.y*(line.pos.y-circle.center.y)));
    ld c = inner(line.pos-circle.center, line.pos-circle.center)-circle.r*circle.r;
```

```
    ld det = b*b-2*a*c;
    ll pred = diff(det, 0);
    if (pred==0)
        result.push_back(line.pos+line.dir*(-b/a));
    else if (pred>0) {
        det = sqrt(det);
        result.push_back(line.pos+line.dir*((-b+det)/a));
        result.push_back(line.pos+line.dir*((-b-det)/a));
    }
    return result;
}
// circle-circle intersection
vector<Point> circle_circle(const Circle& a, const Circle& b) {
    vector<Point> result;
    ll pred = diff(dist(a.center, b.center), a.r+b.r);
    if (pred>0) return result;
    if (pred==0) {
        result.push_back((a.center*b.r+b.center*a.r)*(1/(a.r+b.r)));
        return result;
    }
    ld aa = a.center.x*a.center.x+a.center.y*a.center.y-a.r*a.r;
    ld bb = b.center.x*b.center.x+b.center.y*b.center.y-b.r*b.r;
    ld tmp = (bb-aa)/2.0;
    Point cdiff = b.center-a.center;
    if (diff(cdiff.x, 0)==0) {
        if (diff(cdiff.y, 0)==0)
        return result; // if (diff(a.r, b.r) == 0): same circle
        return circle_line(a, Line{Point{0, tmp/cdiff.y}, Point{1, 0}});
    }
    return circle_line(a,Line{Point{tmp/cdiff.x, 0}, Point{-cdiff.y, cdiff.x}});
}
// circumcircle with three points
Circle circle_from_3pts(const Point& a, const Point& b, const Point& c) {
    Point ba = b-a, cb = c-b;
    Line p{(a+b)*0.5, Point{ba.y, -ba.x}};
    Line q{(b+c)*0.5, Point{cb.y, -cb.x}};
    Circle circle;
    if (!line_line(p, q, circle.center)) circle.r = -1;
```

```
    else  circle.r = dist(circle.center, a);
    return circle;
}
// circumcircle with two points and radius
Circle circle_from_2pts_rad(const Point& a, const Point& b, ld r) {
    ld det = r*r/dist2(a, b)-0.25;
    Circle circle;
    if (det<0)
        circle.r = -1;
    else {
        ld h = sqrt(det);
        // center is to the left of a->b
        circle.center = (a+b)*0.5+Point{a.y-b.y, b.x-a.x}*h;
        circle.r = r;
    }
    return circle;
}
Point inner_center(const Point &a, const Point &b, const Point &c) {
    ld wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
    ld w = wa+wb+wc;
    return { (wa*a.x+wb*b.x+wc*c.x)/w, (wa*a.y+wb*b.y+wc*c.y)/w };
}
Point outer_center(const Point &a, const Point &b, const Point &c) {
    Point d1 = b-a, d2 = c-a;
    ld area = outer(d1, d2);
    ld dx = d1.x*d1.x*d2.y-d2.x*d2.x*d1.y+d1.y*d2.y*(d1.y-d2.y);
    ld dy = d1.y*d1.y*d2.x-d2.y*d2.y*d1.x+d1.x*d2.x*(d1.x-d2.y);
    return { a.x+dx/area/2.0, a.y-dy/area/2.0 };
}
```

Sorting Points

주어진 점들을 반시계방향으로 정렬

```
struct Point {
    ll x, y;
    ll dist2() const {
            return x*x + y*y;
```

```
    }
    ll quadrant() const {
        if (x>=0 && y>=0) return 1;
        if (x<=0 && y>=0) return 2;
        if (x<=0 && y<=0) return 3;
        if (x>=0 && y<=0) return 4;
    }
    bool operator<(const Point& o) const {
        if (quadrant() != o.quadrant()) return quadrant() < o.quadrant();
        ll ccw = x*o.y - y*o.x;
        if (!ccw) return dist2()<o.dist2();
        return 0<ccw;
    }
};
```

Convex Hull & Rotating Calipers (Monotone Chain)

주어진 점들의 외곽을 감싸는 가장 작은 볼록 다각형 (점들에 고무줄 씌운 모양)

```
// Usage: Find the smallest convex polygon that encloses all points
// O(NlogN)
void getConvexHull(vector<Point>& pt, vector<Point>& convex_hull) {
    sort(pt.begin(), pt.end(), [](const Point& a, const Point& b) {
        return a.x==b.x ? a.y<b.y : a.x<b.x;
    });
    vector<Point> up, lo;
    for (const auto& p : pt) {
        while (up.size() >= 2 && ccw(*++up.rbegin(), *up.rbegin(), p) >= 0) u
p.pop_back();
        while (lo.size() >= 2 && ccw(*++lo.rbegin(), *lo.rbegin(), p) <= 0) lo.po
p_back();
        up.push_back(p);
        lo.push_back(p);
    }
    // rotating calipers
    // Usage: Get all antipodal pairs
    // O(N)
    for (ll i=0, j=(ll)lo.size()-1; i+1<up.size() || j > 0;) {
```

```
        get_pair(up[i], lo[j]); // DO WHAT YOU WANT
        if (i + 1 == up.size()) --j;
        else if (j == 0) ++i;
        else if ((up[i+1].y - up[i].y) * (lo[j].x - lo[j-1].x) >
        (up[i+1].x - up[i].x) * (lo[j].y - lo[j-1].y))++i;
        else --j;
    }
    up.insert(up.end(), ++lo.rbegin(), --lo.rend());
    swap(up, convex_hull);
}
```

Point in Polygon Test

점이 다각형 안에 있는 지 판별

```
// Usage: Is pointer in polygon?
// O(N)
bool is_in_polygon(Point p, vector<Point>& poly) {
    ll wn = 0;
    for (ll i=0; i<poly.size(); ++i) {
        ll ni = i+1==poly.size() ? 0 : i+1;
        if (poly[i].y<=p.y) {
            if (poly[ni].y>p.y && ccw(poly[i], poly[ni], p)>0) ++wn;
        }
        else if (poly[ni].y<=p.y && ccw(poly[i], poly[ni], p)<0) --wn;
    }
    return wn != 0;
}
```

Polygon Cut

직선의 왼쪽 기준으로 다각형 자르기

```
// Usage: Get polygon vector left of line
// O(N)
void cut_polygon(Line line, const vector<Point>& polygon, vector<Point>&
cut) {
    if (!polygon.size()) {
```

```
        cut = polygon;
        return;
    }
    using piter = vector<Point>::const_iterator;
    piter la, lan, fi, fip, i, j;
    la = lan = fi = fip = polygon.end();
    i = polygon.end()-1;
    bool lastin = diff(ccw_line(line, polygon[polygon.size()-1]), 0) > 0;
    for (j=polygon.begin(); j!=polygon.end(); j++) {
        bool thisin = diff(ccw_line(line, *j), 0) > 0;
        if (lastin && !thisin)
            la = i, lan = j;
        if (!lastin && thisin)
            fi = j, fip = i;
        i = j;
        lastin = thisin;
    }
    if (fi==polygon.end()) {
        cut = lastin ? polygon : vector<Point>();
        return;
    }
    for (i=fi; i!=lan; i++) {
        if (i == polygon.end()) {
            i = polygon.begin();
            if (i == lan) break;
        }
        cut.push_back(*i);
    }
    Point lc, fc;
    get_cross(Line{ *la, *lan-*la }, line, lc);
    get_cross(Line{ *fip, *fi-*fip }, line, fc);
    cut.push_back(lc);
    if (diff(dist2(lc, fc), 0) != 0) cut.push_back(fc);
}
```

Pick’s Theorem

격자점으로 구성된 simple polygon에 대해 i 는 polygon 내부의 격자수, b 는 polygon 선분 위 격자수, A 는 polygon 넓이라고 할 때, $A = i + b/2 - 1$
