几何

forgottencsc

Sept 19, 2019

二维平面几何

基本定义

点

```
typedef double dbl;
const dbl pi = acos(-1), eps = 1e-7;
dbl sgn(dbl x) { return x < eps ? -1 : x > eps; }

struct vec { dbl x, y; };
vec operator+(vec v1, vec v2) { return { v1.x + v2.x, v1.y + v2.y }; }
vec operator-(vec v1, vec v2) { return { v1.x - v2.x, v1.y - v2.y }; }
dbl operator*(vec v1, vec v2) { return v1.x * v2.x + v1.y * v2.y; }
dbl operator^(vec v1, vec v2) { return v1.x * v2.y - v1.y * v2.x; }
vec operator*(vec v, dbl k) { return { v.x * k, v.y * k }; }
vec operator/(vec v, dbl k) { return { v.x / k, v.y / k }; }

bool operator<(vec v1, vec v2) { return v1.x == v2.x ? v1.y < v2.y : v1.x < v2.x; }
bool operator==(vec v1, vec v2) { return v1.x == v2.x && v1.y == v2.y; }
bool operator>(vec v1, vec v2) { return v2 < v1; }

dbl dot(vec v0, vec v1, vec v2) { return (v1 - v0) * (v2 - v0); }
dbl crx(vec v0, vec v1, vec v2) { return (v1 - v0) ^ (v2 - v0); }
dbl len(vec v) { return hypot(v.x, v.y); }
dbl len2(vec v) { return v * v; }
dbl arg(vec v) { dbl r = atan2(v.y, v.x); return r < 0? 2 * pi + r : r; }

vec unif(vec v) { return v / len(v); }
vec univ(dbl f) { return { cos(f), sin(f) }; }
vec rot(vec p, dbl f) { return { cos(f) * p.x - sin(f) * p.y, sin(f) * p.x + cos(f) * p.y }; }
vec rot(vec o, vec p, vec f) { return o + rot(p - o, f); }
vec rot90(vec p) { return { -p.y, p.x }; }
```

直线

```
struct line { vec p, v; };
vec proj(line l, vec p) { return p + l.v * ((l.p - p) * l.v / len2(l.v)); }
vec refl(vec o, vec p) { return o + o - p; }
vec refl(line l, vec p) { return refl(proj(l, p), p); }
vec litsc(line l1, line l2) { return l2.p + l2.v * ((l1.v ^ (l2.p - l1.p)) / (l2.v ^ l1.v)); }
dbl lpdis(line l, vec p) { return fabs(crx(p, l.p, l.p + l.v)) / len(l.v); }
```

线段

```
struct seg { vec p1, p2; };
bool onseg(seg s, vec p){return!dc(crx(p,s.p1,s.p2))&&dc(dot(p, s.p1, s.p2))==-1;}
```

```cpp
//  0 为不相交，1 为严格相交，2 表示交点为某线段端点，3 为线段平行且部分重合
int sitsc(seg s1, seg s2) {
    vec p1 = s1.p1, p2 = s1.p2, q1 = s2.p1, q2 = s2.p2;
    if (max(p1.x,p2.x)<min(q1.x,q2.x)||min(p1.x,p2.x)>max(q1.x,q2.x)) return 0;
    if (max(p1.y,p2.y)<min(q1.y,q2.y)||min(p1.y,p2.y)>max(q1.y,q2.y)) return 0;
    dbl x=crx(p2,p1,q1),y=crx(p2,p1,q2),z=crx(q2,q1,p1),w=crx(q2,q1,p2);
    if (dc(x)==0&&dc(y)==0) return 3;
    if (dc(x)*dc(y)<0&&dc(z)*dc(w)<0) return 1;
    if (dc(x)*dc(y)<=0&&dc(z)*dc(w)<=0) return 2;
    return 0;
}

dbl spdis(seg s, vec p) {
    if (dot(s.p1, s.p2, p) < eps) return len(p - s.p1);
    if (dot(s.p2, s.p1, p) < eps) return len(p - s.p2);
    return fabs(crx(p, s.p1, s.p2)) / len(s.p1 - s.p2);
}
```

圆

```cpp
struct cir { vec o; dbl r; };

dbl ccarea(cir c1, cir c2) {
    if (c1.r > c2.r) swap(c1, c2);
    dbl d = len(c1.o - c2.o);
    if (dc(d - (c1.r + c2.r)) >=0) return 0;
    if (dc(d - abs(c1.r - c2.r)) <= 0) {
        dbl r = min(c1.r, c2.r);
        return r * r * pi;
    }
    dbl x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d);
    dbl t1 = acos(x / c1.r), t2 = acos((d - x) / c2.r);
    return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r * sin(t1);
}

//  求圆与直线的交点
bool clitsc(cir c, line l, vec& p1, vec& p2) {
    dbl x = l.v * (l.p - c.o), y = l.v * l.v;
    dbl d = x * x - y * ((l.p - c.o) * (l.p - c.o) - c.r * c.r);
    if (dc(d) == -1) return false; d = max(d, 0.);
    vec p = l.p - (x / y) * l.v, w = (sqrt(d) / y) * l.v;
    p1 = p + w; p2 = p - w; return true;
}

//  求圆与圆的交点
bool ccitsc(cir c1, cir c2, vec& p1, vec& p2) {
    //assert(c1 != c2);
    dbl s1 = len(c1.o - c2.o);
    if (dc(s1 - c1.r - c2.r)==1||dc(s1-abs(c1.r-c2.r))==-1)
        return false;
    dbl s2 = (c1.r*c1.r-c2.r*c2.r)/s1, a=(s1+s2)/2, b=(s1-s2)/2;
    vec o = (a/(a+b)) * (c2.o-c1.o) + c1.o;
    vec d = sqrt(c1.r*c1.r-a*a) * rot90(unif(c2.o - c1.o));
    p1 = o + d; p2 = o - d;
    return true;
}

//  求点到圆的切线
bool cptan(cir c, vec p, vec& p1, vec& p2) {
    dbl x = (p-c.o)*(p-c.o), y=x-c.r*c.r;
    if (y < eps) return false;
```

```
    vec o = (c.r*c.r/x)*(p-c.o);
    vec d =rot90((-c.r*sqrt(y)/x)*(p-c.o));
    o = o + c.o; p1 = o + d; p2 = o - d;
    return true;
}

// 求两个圆的外侧公切线
bool ccetan(cir c1, cir c2, line& l1, line& l2) {
    // assert(c1 != c2)
    if (!dc(c1.r - c2.r)) {
        vec v = rot90(c1.r * unif(c2.o - c1.o));
        l1 = { c1.o + v, c2.o - c1.o };
        l2 = { c1.o - v, c2.o - c1.o };
        return true;
    }
    else {
        vec p = (1/(c1.r-c2.r))*(c1.r*c2.o-c2.r*c1.o);
        vec p1, p2, q1, q2;
        if (cptan(c1,p,p1,p2)&&cptan(c2,p,q1,q2)) {
            if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
            l1 = { p1, q1 - p1 }; l2 = { p2, q2 - p2 };
            return true;
        }
    }
    return false;
}

// 求两个圆的内侧公切线
bool ccitan(cir c1, cir c2, line& l1, line& l2) {
    vec p = (1/(c1.r + c2.r)) * (c2.r * c1.o + c1.r * c2.o);
    vec p1, p2, q1, q2;
    if (cptan(c1, p, p1, p2) && cptan(c2, p, q1, q2)) {
        l1 = { p1, q1 - p1 }; l2 = { p2, q2 - p2 };
        return true;
    }
    return false;
}
```

三角形

```
vec incenter(vec a, vec b, vec c) {
    dbl d1 = len(b-c),d2=len(c-a),d3=len(a-b),
        s = fabs(crx(a,b,c));
    return (1/(d1+d2+d3))*(d1*a+d2*b+d3*c);
}

vec circumcenter(vec a, vec b, vec c) {
    b=b-a; c=c-a; dbl d1 = b*b, d2 = c*c, d = 2*(b^c);
    return a - (1/d)*vec{b.y*d2-c.y*d1,c.x*d1-b.x*d2};
}

// To be tested
vec orthocenter(vec a, vec b, vec c) {
    vec ba = b - a, ca = c - a, bc = b - c;
    dbl y = ba.y*ca.y*bc.y,A=ca.x*ba.y-ba.x*ca.y,
        x0=(1/A)*(y+ca.x*ba.y*b.x-ba.x*ca.y*c.x),
        y0=(-ba.x)*(x0-c.x)/ba.y+ca.y;
    return { x0, y0 };
    //return litsc({a,rot90(b-c)},{b,rot90(c-a)});
}
```

多边形

询问点在多边形内

半平面交

```
bool judge(line l0, line l1, line l2) { return sgn((litsc(l1, l2)-l0.p)^l0.v)==1; }
int halfplane_intersection(line* lv, int n, vec* pv) {
    static pair<pair<dbl,dbl>, int> a[N];
    for (int i = 1; i <= n; ++i)
        a[i] = { { arg(lv[i].v), lv[i].p*univ(arg(lv[i].v)-pi/2) }, i };
    sort(a + 1, a + n + 1);
    static int b[N], q[N]; int w = 0, l = 1, r = 0;
    for (int i = 1; i <= n; ++i)
        if (i == 1 || sgn(a[i].first.first-a[i-1].first.first))
            b[++w] = a[i].second;
    for (int i = 1; i <= w; ++i) {
        while (l<r&&judge(lv[b[i]],lv[q[r]],lv[q[r-1]]))--r;
        while (l<r&&judge(lv[b[i]],lv[q[l]],lv[q[l+1]]))++l;
        q[++r]=b[i];
    }
    while(l<r&&judge(lv[q[l]],lv[q[r]],lv[q[r-1]]))--r;
    while(l<r&&judge(lv[q[r]],lv[q[l]],lv[q[l+1]]))++l;
    if (r <= l + 1) return 0;
    int m = 0; q[r+1]=q[l];
    for (int i = l; i <= r; ++i)
        pv[++m]=litsc(lv[q[i]],lv[q[i+1]]);
    return m;
}
```

凸包

按逆时针输出。

```
int convex_hull(vec* p, int n, vec* c) {
    sort(p + 1, p + n + 1); n = unique(p + 1, p + n + 1) - p - 1;
    int m = 0;
    c[1] = p[++m];
    for (int i = 1; i <= n; ++i) {
        while (m > 1 && dc(crx(c[m - 1], c[m], p[i])) != 1) m--;
        c[++m] = p[i];
    }
    int t = m;
    for (int i = n - 1; i; --i) {
        while (m > t && dc(crx(c[m - 1], c[m], p[i])) != 1) m--;
        c[++m] = p[i];
    }
    if (m > 1) m--; c[m + 1] = c[1]; return m;
}
```

闵可夫斯基和

```
int minkowski_sum(vec* cv1, int n1, vec* cv2, int n2, vec* cv) {
    if (n1 == 1 || n2 == 1) {
        if (n1 == 1) swap(n1, n2), swap(cv1, cv2);
        for (int i = 1; i <= n1; ++i)
            cv[i] = cv1[i] + cv2[1];
        return n1;
    }
    static vec dv1[N], dv2[N], dv;
    cv1[n1 + 1] = cv1[1]; cv2[n2 + 1] = cv2[1];
```

```
    for (int i = 1; i <= n1; ++i) dv1[i] = cv1[i + 1] - cv1[i];
    for (int i = 1; i <= n2; ++i) dv2[i] = cv2[i + 1] - cv2[i];
    int m = 0; cv[++m] = cv1[1] + cv2[1];
    int p1 = 1, p2 = 1;
    while (p1 <= n1 || p2 <= n2) {
        if (p1 <= n1 && p2 <= n2)
            dv = (dc((dv1[p1])^(dv2[p2]))!=-1?dv1[p1++]:dv2[p2++]);
        else if (p1 <= n1)
            dv = dv1[p1++];
        else
            dv = dv2[p2++];
        while (m > 1 && !dc((cv[m] - cv[m - 1]) ^ dv)) {
            dv = dv + cv[m] - cv[m - 1];
            m--;
        }
        cv[m + 1] = cv[m] + dv;
        m++;
    }
    if (m > 1) m--; return m;
}
```

询问点在凸包内/点到凸包的切线/叉积最大点 传进 init 的点要求按逆时针排列。

```
typedef pair<dbl, int> pdi;
namespace cvq {
    vec c[N];
    int w, n;

    void init(vec* cv, int m) {
        copy_n(cv + 1, m, c); n = m;
        rotate(c, min_element(c, c + n), c + n);
        c[n] = c[0]; c[n + 1] = c[1];
        w = 0; while (c[w] < c[w + 1]) ++w;
    }

    // 0: 在凸包外, 1: 在凸包上或凸包内
    int contain(vec p) {
        if (p.x < c[0].x || c[w].x < p.x) return false;
        if (crx(c[0], c[w], p) > 0) {
            int e = lower_bound(c + w, c + n + 1, vec{ p.x, inf }, greater<vec>()) - c;
            if (!sgn(p.x - c[e].x)) return p.y <= c[e].y;
            else return crx(c[e - 1], c[e], p) >= 0;
        }
        else {
            int e = lower_bound(c, c + w + 1, vec{ p.x, -inf }) - c;
            if (!sgn(p.x - c[e].x)) return p.y >= c[e].y;
            else return crx(c[e - 1], c[e], p) >= 0;
        }
    }

    pdi crxmax(int l, int r, vec p) {
        int r0 = r;
        while (l <= r) {
            int m = (l + r) >> 1;
            if ((p ^ (c[m + 1] - c[m])) >= 0) l = m + 1;
            else r = m - 1;
        }
        return pdi(p^c[l],l);
    }

    pair<dbl, int> crxmax(vec p) {
        pdi res = sgn(p.x) <= 0 ? crxmax(0, w - 1, p) : crxmax(w, n - 1, p);
```

```cpp
        return max({ res, pdi(p ^ c[0], 0), pdi(p ^ c[w], w) });
    }
    pair<dbl, int> crxmin(vec p) { return crxmax(p * -1); }

    bool ltan(vec p, int i) { return crx(p,c[i],i?c[i-1]:c[n-1])<=0&&crx(p,c[i],c[i+1])<=0; }
    bool rtan(vec p, int i) { return crx(p,c[i],i?c[i-1]:c[n-1])>=0&&crx(p,c[i],c[i+1])>=0; }

    int ltan(int l, int r, vec p) {
        if (ltan(p, r)) return r; r--;
        while (l <= r) {
            int m = (l + r) >> 1;
            if (crx(p,c[m],c[m+1])<0) r = m - 1;
            else l = m + 1;
        }
        return l;
    }

    int rtan(int l, int r, vec p) {
        if (rtan(p, r)) return r; l++;
        while (l <= r) {
            int m = (l + r) >> 1;
            if (crx(p,c[m],m?c[m - 1]:c[n - 1])>0) l = m + 1;
            else r = m - 1;
        }
        return r;
    }

    bool tangent(vec p, int& lp, int& rp) {
        if (contain(p)) return false;
        if (p.x < c[0].x) { lp = ltan(w, n, p); rp = rtan(0, w, p); }
        else if (p.x > c[w].x) { lp = ltan(0, w, p); rp = rtan(w, n, p); }
        else if (crx(c[0], c[w], p) > 0) {
            int e = lower_bound(c + w, c + n + 1, p, greater<vec>()) - c;
            lp = ltan(w, e, p); rp = rtan(e, n, p);
        }
        else {
            int e = lower_bound(c + 0, c + w + 1, p) - c;
            lp = ltan(0, e, p); rp = rtan(e, w, p);
        }
        lp %= n; rp %= n;
        return true;
    }
};
```