典型的Point结构体

struct point {

double x, y;

point(double \_x = 0, double \_y = 0): x(\_x), y(\_y) {

}

void input() {

scanf("%lf%lf", &x, &y);

}

double len() const {

return sqrt(x \* x + y \* y);

}

point trunc(double l) const {

double r = l / len();

return point(x \* r, y \* r);

}

point rotate\_left() const {

return point(-y, x);

}

point rotate\_left(double ang) const {

double c = cos(ang), s = sin(ang);

return point(x \* c - y \* s, y \* c + x \* s);

}

point rotate\_right() const {

return point(y, -x);

}

point rotate\_right(double ang) const {

double c = cos(ang), s = sin(ang);

return point(x \* c + y \* s, y \* c - x \* s);

}

};

bool operator==(const point& p1, const point& p2) {

return sgn(p1.x - p2.x) == 0 && sgn(p1.y - p2.y) == 0;

}

bool operator<(const point& p1, const point& p2) {

return sgn(p1.x - p2.x) == 0 ? sgn(p1.y - p2.y) < 0 : p1.x < p2.x;

}

point operator+(const point& p1, const point& p2) {

return point(p1.x + p2.x, p1.y + p2.y);

}

point operator-(const point& p1, const point& p2) {

return point(p1.x - p2.x, p1.y - p2.y);

}

double operator^(const point& p1, const point& p2) {

return p1.x \* p2.x + p1.y \* p2.y;

}

double operator\*(const point& p1, const point& p2) {

return p1.x \* p2.y - p1.y \* p2.x;

}

void trim(double& d) { d = (d > 1.0 ? 1.0 : d < -1.0 ? -1.0 : d);}

转角法判断点是否在简单多边形的内部

int get\_position(const point& p, const point\* pol, int n) {

double ang = 0;

for (int i = 0; i < n; ++i) {

point p1 = pol[i] - p, p2 = pol[(i + 1) % n] - p;

double c = (p1 ^ p2) / (p1.len() \* p2.len());

trim(c); // 因为精度问题，可能导致本来三角函数值等于1的超过1了

ang += sgn(p1 \* p2) \* acos(c);

}

ang = abs(ang);

return ang < 0.5 \* pi ? -1 : (ang < 1.5 \* pi ? 0 : 1); // -1 在外面， 0 在点上， 1在内部

}

圆与圆的切线

struct circle {

...

vector<pair<point, point> > get\_tangency\_points(const circle& cir) const {

vector<pair<point, point> > t;

double d = (c - cir.c).len();

if (sgn(d - abs(cir.r - r)) <= 0)

return t;

double l = sqrt(abs(d \* d - (cir.r - r) \* (cir.r - r)));

double h1 = r \* l / d, h2 = cir.r \* l / d;

point p = (r > cir.r ? cir.c - c : c - cir.c);

point pp1 = c + p.trunc(sqrt(abs(r \* r - h1 \* h1))), pp2 = cir.c + p.trunc(sqrt(abs(cir.r \* cir.r - h2 \* h2)));

t.push\_back(make\_pair(pp1 + p.rotate\_left().trunc(h1), pp2 + p.rotate\_left().trunc(h2)));

t.push\_back(make\_pair(pp1 - p.rotate\_left().trunc(h1), pp2 - p.rotate\_left().trunc(h2)));

if (sgn(d - (r + cir.r)) <= 0)

return t;

double d1 = d \* r / (r + cir.r), d2 = d \* cir.r / (r + cir.r);

point pp3 = c + (cir.c - c).trunc(r \* r / d1), pp4 = cir.c + (c - cir.c).trunc(cir.r \* cir.r / d2);

double h3 = sqrt(abs(r \* r - (r \* r \* r \* r) / (d1 \* d1))), h4 = sqrt(abs(cir.r \* cir.r - (cir.r \* cir.r \* cir.r \* cir.r) / (d2 \* d2)));

t.push\_back(make\_pair(pp3 + (cir.c - c).rotate\_left().trunc(h3), pp4 + (c - cir.c).rotate\_left().trunc(h4)));

t.push\_back(make\_pair(pp3 - (cir.c - c).rotate\_left().trunc(h3), pp4 - (c - cir.c).rotate\_left().trunc(h4)));

return t;

}

};

凸包

int dn, hd[max\_n], un, hu[max\_n];

void get\_convex\_hull(point\* p, int n, point\* pol, int& m) {

sort(p, p + n);

dn = un = 2;

hd[0] = hu[0] = 0;

hd[1] = hu[1] = 1;

for (int i = 2; i < n; ++i) {

for (; dn > 1 && sgn((p[hd[dn - 1]] - p[hd[dn - 2]]) \* (p[i] - p[hd[dn - 1]])) <= 0; --dn);

for (; un > 1 && sgn((p[hu[un - 1]] - p[hu[un - 2]]) \* (p[i] - p[hu[un - 1]])) >= 0; --un);

hd[dn++] = hu[un++] = i;

}

m = 0;

for (int i = 0; i < dn - 1; ++i)

pol[m++] = p[hd[i]];

for (int i = un - 1; i > 0; --i)

pol[m++] = p[hu[i]];

}

三维几何中点到直线、平面的距离

double get\_distance(const point& p, const point& p1, const point& p2) {

return abs(((p1 - p) \* (p2 - p)).len() / (p1 - p2).len());

}

double get\_distance(const point& p, const point& p1, const point& p2, const point& p3) {

point v = (p2 - p1) \* (p3 - p1);

return abs((v ^ (p - p1)) / v.len());

}

===================ixay================

const double pi = acos(-1.);

const double eps = 1e-9;

int sgn(double x) {

return (x > eps) - (x < -eps);

}

struct P {

double x, y;

P() {}

P(double \_x, double \_y): x(\_x), y(\_y) {}

P operator + (const P &a) const {

return P(x + a.x, y + a.y);

}

P operator - (const P &a) const {

return P(x - a.x, y - a.y);

}

P operator \* (const double &m) const {

return P(x \* m, y \* m);

}

P operator / (const double &m) const {

return P(x / m, y / m);

}

P set(const double &m) const {

double len = length();

return P(x \* m / len, y \* m / len);

}

P turn(const double &m) const {

double c = cos(m), s = sin(m);

return P(x \* c - y \* s, x \* s + y \* c);

}

double length() const {

return sqrt(x \* x + y \* y);

}

double dist(const P &a) const {

return sqrt(SQR(x - a.x) + SQR(y - a.y));

}

double cross(const P &a, const P &b) const {

return (a.x - x) \* (b.y - y) - (a.y - y) \* (b.x - x);

}

double cross(const P &a) const {

return x \* a.y - y \* a.x;

}

double dot(const P &a, const P &b) {

return (a.x - x) \* (b.x - x) + (a.y - y) \* (b.y - y);

}

void input() {

scanf("%lf%lf", &x, &y);

}

void output() const {

printf("(%lf, %lf)\n", x, y);

}

};

point\_in\_polygon

//判断点在多边形内，多边形上的点也作为多边形内

int find(const P &t) {

return t.x >= 0 ? (t.y >= 0 ? 0 : 3) : (t.y >= 0 ? 1 : 2);

}

// 0 表示点在多边形外，1 表示点在多边形上, 2 表示点在多边形内

int points\_in\_polygon(const P &p, const vector<P> &pl) {

int n = pl.size(), sum = 0;

int t1 = find(pl.back() - p);

for (int i = 0; i < n; ++i) {

double f = p.cross(pl[(i + n - 1) % n], pl[i]);

if (sgn(f) == 0 && sgn(p.dot(pl[(i + n - 1) % n], pl[i])) <= 0) return 1;

int t2 = find(pl[i] - p);

if (t2 == (t1 + 1) % 4) ++sum;

else if (t2 == (t1 + 3) % 4) --sum;

else if (t2 == (t1 + 2) % 4) {

if (f > 0) sum += 2;

else sum -= 2;

}

t1 = t2;

}

if (sum) return 2;

return 0;

}