目录

[1 2D基础几何 1](#_Toc525171200)

[2 直线和圆 2](#_Toc525171201)

[3 点在多边形内判相交 4](#_Toc525171202)

[4 2D凸包相关 4](#_Toc525171203)

[5 半平面交 6](#_Toc525171204)

[6 圆面积相关 7](#_Toc525171205)

[7 平面划分 9](#_Toc525171206)

[8 基础3D几何 10](#_Toc525171207)

[9 凸包3D 11](#_Toc525171208)

[10 求空间点到某平面的投影点 13](#_Toc525171209)

1 2D基础几何

const double eps = 1e-10 , pi = acos(-1.0);

inline int dcmp(double x) {

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

}

struct Point {

double x , y;

Point (double x = 0 , double y = 0) : x(x) , y(y) {}

void input() {

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

}

bool operator < (const Point& R) const {

if (dcmp(x - R.x) == 0)

return dcmp(y - R.y) < 0;

return dcmp(x - R.x) < 0;

}

bool operator == (const Point& R) const {

return dcmp(x - R.x) == 0 && dcmp(y - R.y) == 0;

}

Point operator + (const Point& R) const {

return Point(x + R.x , y + R.y);

}

Point operator - (const Point& R) const {

return Point(x - R.x , y - R.y);

}

Point operator \* (const double& R) const {

return Point(x \* R , y \* R);

}

Point operator / (const double& R) const {

return Point(x / R , y / R);

}

double operator ^ (const Point& R) const {

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

}

double operator % (const Point& R) const {

return x \* R.x + y \* R.y;

}

double len() {

return sqrt(\*this % \*this);

}

double angle() {

return atan2(y , x);

}

};

// 两个向量的夹角，不分正负[0,pi)

double Angle(Point A , Point B) {

return acos((A % B) / A.len() / B.len());

}

// 逆时针旋转

Point Rotate(Point A , double rad) {

double Sin = sin(rad) , Cos = cos(rad);

return Point(A.x \* Cos - A.y \* Sin , A.x \* Sin + A.y \* Cos);

}

// 向量的单位法向量，利用旋转得到

Point Normal(Point A) {

double L = A.len();

return Point(-A.y / L , A.x / L);

}

// 直线交点，v和w为两个直线的方向向量，

// 设交点的参数为P+vt,Q+wt,连立方程解t

// 线段，射线对这个t的参数有限制，很好理解。

Point GetLineIntersection(Point P , Point v , Point Q , Point w) {

Point u = P - Q;

double t1 = (w ^ u) / (v ^ w);

return P + v \* t1;

}

// 点到直线有向距离，这里直线是用两个点表示的

double DistancePointToLine(Point P , Point A , Point B) {

Point v = B - A;

return (v ^ (P - A)) / v.len();

}

// 点到线段距离，就是上面的代码判断一下P在AB上投影的位置。

double DistancePointToSegment(Point P , Point A , Point B) {

if (A == B) return (P - A).len();

Point v1 = B - A , v2 = P - A , v3 = P - B;

if (dcmp(v1 % v2) < 0) return v2.len();

if (dcmp(v1 % v3) > 0) return v3.len();

return fabs(v1 ^ v2) / v1.len();

}

// 返回点在直线上的投影

Point GetLineProjection(Point P , Point A , Point B) {

Point v = B - A;

return A + v \* (v % (P - A) / (v % v));

}

// 判断线段是否严格相交。

bool SegmentProperIntersection(Point a1 , Point a2 , Point b1 , Point b2) {

double c1 = (a2 - a1) ^ (b1 - a1);

double c2 = (a2 - a1) ^ (b2 - a1);

if (dcmp(c1) == 0 && dcmp(c2) == 0) {

if (a2 < a1) swap(a1 , a2);

if (b2 < b1) swap(b1 , b2);

return max(a1 , b1) < min(a2 , b2);

}

double c3 = (b2 - b1) ^ (a1 - b1);

double c4 = (b2 - b1) ^ (a2 - b1);

return dcmp(c1) \* dcmp(c2) < 0 && dcmp(c3) \* dcmp(c4) < 0;

}

// 点是否在线段上,判定方式为到两个端点的方向是否不一致。

bool OnSegment(Point P , Point a1 , Point a2) {

double len = (P - a1).len();

if (dcmp(len) == 0) return true;

a1 = a1 - P , a2 = a2 - P;

return dcmp((a1 ^ a2) / len) == 0 && dcmp(a1 % a2) <= 0;

}

2 直线和圆

struct Line {

Point P , V; // P + Vt

double angle;

Line () {}

Line (Point A , Point B) {

P = A , V = B - A;

angle = atan2(V.y , V.x);

}

bool operator < (const Line& R) const {

return angle < R.angle;

}

Point point(double t) {

return P + V \* t;

}

};

struct Circle {

Point O;

double r;

Circle () {}

Circle (Point \_O , double \_r) {

O = \_O , r = \_r;

}

Point point(double arc) {

return Point(O.x + cos(arc) \* r , O.y + sin(arc) \* r);

}

void input() {

O.input() , scanf("%lf",&r);

}

};

// 判定直线与圆相交

// 方法为连立直线的参数方程与圆的方程，很好理解

// t1,t2为两个参数，sol为点集。有了参数，射线线段什么的也很方便

int getLineCircleIntersection(Line L , Circle C , double& t1 , double& t2 , vector<Point>& sol) {

double a = L.V.x , b = L.P.x - C.O.x , c = L.V.y , d = L.P.y - C.O.y;

double e = a \* a + c \* c , f = 2 \* (a \* b + c \* d);

double g = b \* b + d \* d - C.r \* C.r;

double delta = f \* f - 4 \* e \* g;

if (dcmp(delta) < 0) return 0;

if (dcmp(delta) == 0) {

t1 = t2 = -f / (2 \* e);

sol.push\_back(L.point(t1));

return 1;

}

t1 = (-f - sqrt(delta)) / (e + e);

t2 = (-f + sqrt(delta)) / (e + e);

sol.push\_back(L.point(t1)) , sol.push\_back(L.point(t2));

return 2;

}

// 判定圆和圆之间的关系

// 内含，内切，相交，重合，外切，相离

int getCircleCircleIntersection(Circle C1 , Circle C2 , vector<Point>& sol) {

double d = (C1.O - C2.O).len();

if (dcmp(d) == 0) { //同心

if (dcmp(C1.r - C2.r) == 0)//重合

return -1;

return 0;//内含

}

if (dcmp(C1.r + C2.r - d) < 0) return 0;//相离

if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;//内含

double a = (C2.O - C1.O).angle();

double p = (C1.r \* C1.r + d \* d - C2.r \* C2.r) / (2 \* C1.r \* d);

p = max(-1.0 , min(1.0 , p));

double da = acos(p);

Point P1 = C1.point(a - da) , P2 = C1.point(a + da);

sol.push\_back(P1);

if (dcmp(da) == 0) return 1; //切

sol.push\_back(P2);

return 2;

}

// 过点p到圆C的切线。返回切线条数，sol里为方向向量

int getTangents(Point P, Circle C, vector<Point>& sol) {

Point u = C.O - P;

double dist = u.len();

if(dist < C.r) return 0;

if(dcmp(dist - C.r) == 0) {

sol.push\_back(Rotate(u, pi / 2));

return 1;

} else {

double ang = asin(C.r / dist);

sol.push\_back(Rotate(u, +ang));

sol.push\_back(Rotate(u, -ang));

return 2;

}

}

//两个圆的公切线，对应切点存在ab里面

int getTangents(Circle A , Circle B , Point\* a , Point\* b) {

int cnt = 0;

if (A.r < B.r) swap(A , B) , swap(a , b);

double dist = (A.O - B.O).len() , dr = A.r - B.r , sr = A.r + B.r;

if (dcmp(dist - dr) < 0) // 内含

return 0;

double base = (B.O - A.O).angle();

if (dcmp(dist) == 0 && dcmp(A.r - B.r) == 0)

return -1;//重合

if (dcmp(dist - dr) == 0) {//内切

a[cnt] = A.point(base);

b[cnt] = B.point(base);

return 1;

}

double ang = acos(dr / dist);//非上述情况，两条外公切线

a[cnt] = A.point(base + ang) , b[cnt] = B.point(base + ang) , ++ cnt;

a[cnt] = A.point(base - ang) , b[cnt] = B.point(base - ang) , ++ cnt;

if (dcmp(dist - sr) == 0) {// 外切，中间一条内公切线

a[cnt] = A.point(base) , b[cnt] = B.point(pi + base) , ++ cnt;

} else if (dcmp(dist - sr) > 0) {

ang = acos(sr / dist);//相离，两条内公切线

a[cnt] = A.point(base + ang) , b[cnt] = B.point(pi + base + ang) , ++ cnt;

a[cnt] = A.point(base - ang) , b[cnt] = B.point(pi + base - ang) , ++ cnt;

}

return cnt;

}

// 外接圆，三根中线交点

Circle CircumscribedCircle(Point A , Point B , Point C) {

Point D = (B + C) / 2 , d = Normal(B - C);

Point E = (A + C) / 2 , e = Normal(A - C);

Point P = GetLineIntersection(D , d , E , e);

return Circle(P , (C - P).len());

}

// 内接圆，黑科技

Circle InscribedCircle(Point A , Point B , Point C) {

double a = (B - C).len() , b = (A - C).len() , c = (A - B).len();

Point P = (A \* a + B \* b + C \* c) / (a + b + c);

return Circle(P , fabs(DistancePointToLine(P , A , B)));

}

3 点在多边形内判相交

bool pointInPolygon(Point P , Point \*p , int n) {

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

if (OnSegment(P , p[i] , p[i + 1]))

return 0;

int res = 0;

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

Point a = p[i] , b = p[i + 1];

if (a.y > b.y) swap(a , b);

if (dcmp((a - P) ^ (b - P)) < 0 && dcmp(a.y - P.y) < 0 && dcmp(b.y - P.y) >= 0)

res ^= 1;

}

return res;

}

4 2D凸包相关

inline LL OnLeft(Point P , Point A , Point B) {

return (B - A) ^ (P - A);

}

/\*\*\*\*\*\*\*\*\* Naive 凸包 2.0 O(n+m) \*\*\*\*\*\*\*\*\*/

int top = 0;

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

while (top > 1 && OnLeft(p[i] , s[top - 2] , s[top - 1]) <= 0) {

-- top;

}

s[top ++] = p[i];

}

int tmp = top;

for (int i = n - 2 ; i >= 0 ; -- i) {

while (top > tmp && OnLeft(p[i] , s[top - 2] , s[top - 1]) <= 0) {

-- top;

}

s[top ++] = p[i];

}

if (n > 1)

-- top;

/\*\*\*\*\*\*\*\*\* Minkowski-Sum O(n+m) \*\*\*\*\*\*\*\*\*/

Vec.clear();

Point cur = a[0] + b[0];

for (int i = 0 , j = 0 ; i < n || j < m ; ) {

if (i < n && (j == m || ((a[i + 1] - a[i]) ^ (b[j + 1] - b[j])) >= 0)) {

cur = cur + a[i + 1] - a[i];

++ i;

} else {

cur = cur + b[j + 1] - b[j];

++ j;

}

Vec.push\_back(make\_pair(cur , 1));

}

/\*\*\*\*\*\*\* 点在凸多边形内判定 O(logn) \*\*\*\*\*\*\*/

bool InConvex(Point q) {

if (OnLeft(q , p[0] , p[1]) < 0 || OnLeft(q , p[0] , p[n - 1]) > 0)

return 0;

int l = 2 , r = n - 1;

while (l < r) {

int mid = l + r >> 1;

if (OnLeft(q , p[0] , p[mid]) <= 0) {

r = mid;

} else {

l = mid + 1;

}

}

return OnLeft(q , p[r - 1] , p[r]) >= 0;

}

/\*\*\*\*\*\*\* 点到凸多边形的切线 O(logn) \*\*\*\*\*\*\*/

#define above(b , c) (OnLeft(b , q , c) > 0)

#define below(b , c) (OnLeft(b , q , c) < 0)

int getRtangent(Point q) { // find max

int ret = 0;

int l = 1 , r = n - 1;

while (l <= r) {

int dnl = above(p[l] , p[l + 1]);

int mid = l + r >> 1;

int dnm = above(p[mid] , p[mid + 1]);

if (dnm) {

if (above(p[mid], p[ret])) {

ret = mid;

}

}

if (dnl) {

if (above(p[l], p[ret])) {

ret = l;

}

if (dnm && above(p[mid] , p[l])) {

r = mid - 1;

} else {

l = mid + 1;

}

} else {

if (!dnm && above(p[mid] , p[l])) {

l = mid + 1;

} else {

r = mid - 1;

}

}

}

return ret;

}

int getLtangent(Point q) { // find min

int ret = 0;

int l = 1 , r = n - 1;

while (l <= r) {

int dnl = below(p[l] , p[l - 1]);

int mid = l + r + 1 >> 1;

int dnm = below(p[mid] , p[mid - 1]);

if (dnm) {

if (below(p[mid], p[ret])) {

ret = mid;

}

}

if (dnl) {

if (below(p[l], p[ret])) {

ret = l;

}

if (dnm && below(p[mid] , p[l])) {

l = mid + 1;

} else {

r = mid - 1;

}

} else {

if (!dnm && below(p[mid] , p[l])) {

r = mid - 1;

} else {

l = mid + 1;

}

}

}

return ret;

}

/\*\*\*\*\*\* 直线对凸多边形的交点 O(logn) \*\*\*\*\*\*/

double arc[N] , sum[N];

void init() {

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

p[i + n] = p[i];

} p[n + n] = p[0];

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

sum[i + 1] = sum[i] + (p[i] ^ p[i + 1]);

}

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

int j = (i + 1) % n;

arc[i] = atan2(p[j].y - p[i].y , p[j].x - p[i].x);

if (i && arc[i] < arc[i - 1]) {

arc[i] += pi + pi;

}

}

}

int getseg(Point P , Point V , int l , int r) {

-- l;

while (l < r) {

int mid = l + r + 1 >> 1;

if ((V ^ (p[mid] - P)) < 0) {

l = mid;

} else {

r = mid - 1;

}

}

return l;

}

void work(Point A , Point B) {

if (B < A) {

swap(A , B);

}

double al = atan2(B.y - A.y , B.x - A.x);

if (al < arc[0]) al += pi + pi;

int Left = (lower\_bound(arc , arc + n , al) - arc) % n;

double ar = atan2(A.y - B.y , A.x - B.x);

if (ar < arc[0]) ar += pi + pi;

int Right = lower\_bound(arc , arc + n , ar) - arc;

int down = getseg(A , B - A , Left , Right);

int up = getseg(B , A - B , Right , Left + n);

if (down < Left || up < Right) {

puts("0.000000");

} else {

Point D = GetLineIntersection(A , B - A , p[down] , p[down + 1] - p[down]);

Point U = GetLineIntersection(B , A - B , p[up] , p[up + 1] - p[up]);

//printf("%f %f / %f %f\n" , D.x , D.y , U.x , U.y);

double area = (D ^ p[down + 1]) + (sum[up] - sum[down + 1]) + (p[up] ^ U) + (U ^ D);

printf("%.6f\n" , min(sum[n] - area , area) / 2);

}

}

5 半平面交

typedef vector<Point> Polygon;

//用有向直线AB的左半平面切割 O(n)

Polygon CutPolygon(const Polygon& poly , Point A , Point B) {

Polygon newpoly;

int n = poly.size();

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

const Point &C = poly[i] , &D = poly[(i + 1) % n];

if (dcmp((B - A) ^ (C - A)) >= 0)

newpoly.push\_back(C);

if (dcmp((B - A) ^ (C - D)) != 0) {

double t = ((B - A) ^ (C - A)) / ((D - C) ^ (B - A));

if (dcmp(t) > 0 && dcmp(t - 1) < 0)

newpoly.push\_back(C + (D - C) \* t);

}

}

return newpoly;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline bool Onleft(Line L , Point P) {

return (L.V ^ (P - L.P)) > 0;

}

Point GetLineIntersection(Line A , Line B) {

Point u = A.P - B.P;

double t = (B.V ^ u) / (A.V ^ B.V);

return A.point(t);

}

Point p[N];

Line q[N];

int HalfPlaneIntersection(Line\* L , int n , Point\* Poly) {

sort(L , L + n);

int top = 0 , bot = 0;

q[0] = L[0];

for (int i = 1 ; i < n ; ++ i) {

while (top < bot && !Onleft(L[i] , p[bot - 1])) -- bot;

while (top < bot && !Onleft(L[i] , p[top])) ++ top;

q[++ bot] = L[i];

if (dcmp(L[i].V ^ q[bot - 1].V) == 0) {

-- bot;

if (Onleft(q[bot] , L[i].P))

q[bot] = L[i];

}

if (top < bot)

p[bot - 1] = GetLineIntersection(q[bot - 1] , q[bot]);

}

while (top < bot && !Onleft(q[top] , p[bot - 1])) -- bot;

if (bot - top <= 1) return 0;

p[bot] = GetLineIntersection(q[bot] , q[top]);

int m = 0;

for (int i = top ; i <= bot ; ++ i) Poly[m ++] = p[i];

return m;

}

6 圆面积相关

/\*\*\*\*\*\*圆和多边形求交\*\*\*\*/

double sector\_area(Point A , Point B , double R) {

double theta = Angle(A) - Angle(B);

while (theta < 0) theta += pi + pi;

while (theta >= pi + pi) theta -= pi + pi;

theta = min(theta , pi + pi - theta);

return R \* R \* theta;

}//a[n] = a[0]

double cal(double R) {

double area = 0;

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

double t1 = 0 , t2 = 0 , delta;

Line L = Line(a[i] , a[i + 1]);

int cnt = getLineCircleIntersection(L , Circle(Point(0 , 0) , R) , t1 , t2);

Point X = L.point(t1) , Y = L.point(t2);

bool f1 = dcmp(a[i].len() - R) <= 0 , f2 = dcmp(a[i + 1].len() - R) <= 0;

if (f1 && f2)

delta = fabs(a[i] ^ a[i + 1]);

else if (!f1 && f2) {

delta = sector\_area(a[i] , X , R) + fabs(X ^ a[i + 1]);

} else if (f1 && !f2) {

delta = fabs(a[i] ^ Y) + sector\_area(Y , a[i + 1] , R);

} else {

if (cnt > 1 && 0 < t1 && t1 < 1 && 0 < t2 && t2 < 1) {

delta = sector\_area(a[i] , X , R) + sector\_area(Y , a[i + 1] , R) + fabs(X ^ Y);

} else {

delta = sector\_area(a[i] , a[i + 1] , R);

}

}

area += delta \* dcmp(a[i] ^ a[i + 1]);

}

return area / 2;

}

/\*\*\*\*\*\*\*\*\*圆交/并\*\*\*\*\*\*\*/

void getarea() { // 计算圆并的重心，必要的时候可以去除有包含关系的圆

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

vector< pair<double , int> > Vec;

int cnt = 1;

Vec.push\_back({0 , 0});

Vec.push\_back({2 \* pi , 0});

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

double dist = (c[j].O - c[i].O).len();

if (dcmp(dist) == 0 && dcmp(c[i].r - c[j].r) == 0) {

if (i < j) {

++ cnt;

}

continue;

}

if (dcmp(dist - c[j].r - c[i].r) >= 0) {

continue;

}

if (dcmp(dist + c[j].r - c[i].r) <= 0) { // j in i

continue;

}

if (dcmp(dist + c[i].r - c[j].r) <= 0) { // i in j

++ cnt;

continue;

}

double an = atan2(c[j].O.y - c[i].O.y , c[j].O.x - c[i].O.x);

double p = (c[i].r \* c[i].r + dist \* dist - c[j].r \* c[j].r) / (2 \* c[i].r \* dist);

double da = acos(max(-1.0 , min(1.0 , p)));

double L = an - da , R = an + da;

//printf("%d : %f %f\n" , j , L , R);

if (L < 0) L += 2 \* pi;

if (R < 0) R += 2 \* pi;

if (L >= 2 \* pi) L -= 2 \* pi;

if (R >= 2 \* pi) R -= 2 \* pi;

Vec.push\_back({L , 1});

Vec.push\_back({R , -1});

if (L >= R) {

++ cnt;

}

}

sort(Vec.begin() , Vec.end());

for (int j = 0 ; j + 1 < Vec.size() ; ++ j) {

//printf("%d : %d %f\n" , j , cnt , Vec[j].first);

cnt += Vec[j].second;

if (cnt == 1) {

double delta = Vec[j + 1].first - Vec[j].first;

if (dcmp(delta) <= 0)

continue;

double SIN = sin(delta / 2);

Point W = Point(0 , 4 \* c[i].r \* SIN \* SIN \* SIN / (3 \* (delta - sin(delta))));

W = Rotate(W , (Vec[j + 1].first + Vec[j].first - pi) / 2) + c[i].O;

double area = c[i].r \* c[i].r \* (delta - sin(delta));

sx -= area \* W.x;

sy -= area \* W.y;

s -= area;

Point A = c[i].point(Vec[j].first) , B = c[i].point(Vec[j + 1].first);

area = (A ^ B);

sx -= area \* (A.x + B.x) / 3;

sy -= area \* (A.y + B.y) / 3;

s -= area;

}

}

}

}

7 平面划分

void work() {

scanf("%d" , &n);

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

L[i].input();

P[i] = L[i];

}

int m = n;

for (int i = 0 ; i + 1 < n ; ++ i)

for (int j = i + 1 ; j + 1 < n ; ++ j) {

if (dcmp((P[i + 1] - P[i]) ^ (P[j + 1] - P[j])) != 0)

P[m ++] = GetLineIntersection(P[i] , P[i + 1] - P[i] , P[j] , P[j + 1] - P[j]);

}

sort(P , P + m);

m = unique(P , P + m) - P;

memset(pre , -1 , sizeof(pre));

set< pair<int , int> > Hash;

for (int i = 0 ; i + 1 < n ; ++ i) {

vector< pair <Point , int> > V;

for (int j = 0 ; j < m ; ++ j)

if (OnSegment(P[j] , L[i] , L[i + 1]))

V.push\_back(make\_pair(P[j] , j));

sort(V.begin() , V.end());

for (int j = 0 ; j + 1 < V.size() ; ++ j) {

int x = V[j].second , y = V[j + 1].second;

if (!Hash.count(make\_pair(x , y))) {

Hash.insert(make\_pair(x , y));

e[mcnt] = (edge) {y , pre[x]} , pre[x] = mcnt ++;

}

if (!Hash.count(make\_pair(y , x))) {

Hash.insert(make\_pair(y , x));

e[mcnt] = (edge) {x , pre[y]} , pre[y] = mcnt ++;

}

}

}

for (int x = 0 ; x < m ; ++ x) {

vector< pair<double , int> > V;

for (int i = pre[x] ; ~i ; i = e[i].next) {

int y = e[i].x;

V.push\_back(make\_pair((P[y] - P[x]).arg() , i));

}

sort(V.begin() , V.end());

for (int i = 0 ; i < V.size() ; ++ i) {

int j = (i + 1) % V.size();

Next[V[j].second ^ 1] = V[i].second;

}

}

double res = 0;

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

if (!vis[i]) {

int x = i;

double area = 0;

while (!vis[x]) {

vis[x] = 1;

area += (P[e[x ^ 1].x] ^ P[e[x].x]);

x = Next[x];

}

if (x == i && dcmp(area) > 0)

res += area;

}

}

printf("%.8f\n" , res / 2);

}

8 基础3D几何

const double eps = 1e-8 , pi = acos(-1.0);

inline int dcmp(double x) {

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

}

struct Point {

double x , y , z;

Point () {x = y = z = 0;}

Point (double \_x , double \_y , double \_z) {

x = \_x , y = \_y , z = \_z;

}

void input() {

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

}

bool operator < (const Point &R) const {

if (dcmp(x - R.x) != 0)

return x < R.x;

if (dcmp(y - R.y) != 0)

return y < R.y;

return z < R.z;

}

bool operator == (const Point &R) const {

return dcmp(x - R.x) == 0 && dcmp(y - R.y) == 0 && dcmp(z - R.z) == 0;

}

Point operator + (const Point& R) const {

return Point(x + R.x , y + R.y , z + R.z);

}

Point operator - (const Point& R) const {

return Point(x - R.x , y - R.y , z - R.z);

}

Point operator \* (const double& R) const {

return Point(x \* R , y \* R , z \* R);

}

Point operator / (const double& R) const {

return Point(x / R , y / R , z / R);

}

double operator % (const Point& R) const {

return x \* R.x + y \* R.y + z \* R.z;

}

Point operator ^ (const Point& R) const {

return Point(y \* R.z - z \* R.y , z \* R.x - x \* R.z , x \* R.y - y \* R.x);

}

inline double len() {

return sqrt(\*this % \*this);

}

};

Point GetLinePlaneProjection(Point A , Point P , Point n) {

double t = (n % (P - A)) / (n % n);

return A + n \* t; // t \* n.len() 是距离

} // 直线平面投影

Point GetLinePlaneIntersection(Point A , Point V , Point P , Point n) {

double t = (n % (P - A)) / (n % V);

return A + V \* t;

} // 直线平面交点

inline double area(Point A , Point B , Point C) {

return ((B - A) ^ (C - A)).len();

}

bool PointinTri(Point P) {

double area1 = area(P , a[0] , a[1]);

double area2 = area(P , a[1] , a[2]);

double area3 = area(P , a[2] , a[0]);

return dcmp(area1 + area2 + area3 - area(a[0] , a[1] , a[2])) == 0;

}

double GetLineIntersection(Point P , Point v , Point Q , Point w) {

//共面时使用

Point u = P - Q;

Point delta = v ^ w , cross = w ^ u;

if (dcmp(delta.z) != 0)

return cross.z / delta.z;

else if (dcmp(delta.y) != 0)

return cross.y / delta.y;

else if (dcmp(delta.x) != 0)

return cross.x / delta.x;

else {

return 1e60;

}

}

//a点绕Ob向量逆时针旋转弧度angle. cossin可预先计算

Point Rotate(Point a, Point b, double angle) {

static Point e1 ,e2 , e3;

b = b / b.len() , e3 = b;

double lens = a % e3;

e1 = a - e3 \* lens;

if (dcmp(e1.len()) > 0)

e1 = e1 / e1.len();

else

return a;

e2 = e1 ^ e3;

double x1 = a % e2 , y1 = a % e1 , x2 , y2;

x2 = x1 \* cos(angle) - y1 \* sin(angle);

y2 = x1 \* sin(angle) + y1 \* cos(angle);

return e3 \* lens + e1 \* y2 + e2 \* x2;

}

/\*\*

绕任意轴（过原点）逆时针旋转（注意要把轴向量归一化，不然会在“点在轴上”这个情况下出问题）

rotate x y z d

| (1-cos(d))\*x\*x+cos(d) (1-cos(d))\*x\*y+sin(d)\*z (1-cos(d))\*x\*z-sin(d)\*y 0 |

| (1-cos(d))\*y\*x-sin(d)\*z (1-cos(d))\*y\*y+cos(d) (1-cos(d))\*y\*z+sin(d)\*x 0 |

| (1-cos(d))\*z\*x+sin(d)\*y (1-cos(d))\*z\*y-sin(d)\*x (1-cos(d))\*z\*z+cos(d) 0 |

| 0 0 0 1 |

\*\*/

9 凸包3D

double mix(const Point &a, const Point &b, const Point &c) {

return a % (b ^ c);

}

const int N = 305;

int mark[N][N];

Point info[N];

int n , cnt;

double area(int a, int b, int c) {

return ((info[b] - info[a]) ^ (info[c] - info[a])).len();

}

double volume(int a, int b, int c, int d) {

return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]);

}

struct Face {

int v[3];

Face() {}

Face(int a, int b, int c) {

v[0] = a , v[1] = b , v[2] = c;

}

int& operator [] (int k) {

return v[k];

}

};

vector <Face> face;

inline void insert(int a, int b, int c) {

face.push\_back(Face(a, b, c));

}

void add(int v) {

vector <Face> tmp;

int a, b, c;

cnt ++;

for (int i = 0; i < face.size() ; ++ i) {

a = face[i][0] , b = face[i][1] , c = face[i][2];

if (dcmp(volume(v, a, b, c)) < 0)

mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[c][a] = mark[a][c] = cnt;

else

tmp.push\_back(face[i]);

}

face = tmp;

for (int i = 0; i < tmp.size() ; ++ i) {

a = face[i][0] , b = face[i][1] , c = face[i][2];

if (mark[a][b] == cnt) insert(b, a, v);

if (mark[b][c] == cnt) insert(c, b, v);

if (mark[c][a] == cnt) insert(a, c, v);

}

}

int Find() {

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

Point ndir = (info[0] - info[i]) ^ (info[1] - info[i]);

if (ndir == Point())

continue;

swap(info[i], info[2]);

for (int j = i + 1; j < n; j++)

if (dcmp(volume(0, 1, 2, j)) != 0) {

swap(info[j], info[3]);

insert(0, 1, 2);

insert(0, 2, 1);

return 1;

}

}

return 0;

}

void work() {

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

info[i].input();

sort(info, info + n);

n = unique(info, info + n) - info;

face.clear();

random\_shuffle(info, info + n);

if (Find()) {

memset(mark, 0, sizeof(mark));

cnt = 0;

for (int i = 3; i < n; ++ i) add(i);

vector<Point> Ndir;

for (int i = 0; i < face.size() ; ++i) {

Point p = (info[face[i][0]] - info[face[i][1]]) ^ (info[face[i][2]] - info[face[i][1]]);

p = p / p.len();

Ndir.push\_back(p);

}

sort(Ndir.begin(), Ndir.end());

int ans = unique(Ndir.begin(), Ndir.end()) - Ndir.begin();

printf("%d\n", ans);

} else {

printf("1\n");

}

}

10 求空间点到某平面的投影点

/\*

计算空间点到平面的投影点坐标

0、p为平面外任意一点；

1、pp为所求的投影点坐标；

2、A为平面上任意已知点；

3、n为平面上的法线；

n的计算方法：

一般会已知平面上两个以上的点坐标，例如我是为了求点在任意三角形上的投影点，我当然会

知道三角形的三个点坐标，通过其中两个点坐标可以求出法向量n。

假设已知平面为三角形，其三个顶点分别为A（x,y,z）,B(x,y,z), C(x,y,z).

AB = (Bx-Ax,By-Ay,Bz-Az);AB为向量；

AC = (Cx-Ax,Cy-Ay,Cz-Az);AC为向量；

n为法向量

n = AB X AC

叉积公式：（对应版子中的^运算）

=>nx = ABy\*ACz-ABz\*ACy

ny = ABz\*ACx-ABx\*ACz

nz = ABx\*ACy-ABy\*ACx

注意：以上的Ax是A的x坐标；

ABx指的是AB向量的x分量

\*/

point pro(point p)

{

point pp;

pp.x = (n.x\*n.y\*A.y + n.y\*n.y\*p.x - n.x\*n.y\*p.y + n.x\*n.z\*A.z + n.z\*n.z\*p.x - \

n.x\*n.z\*p.z + n.x\*n.x\*A.x) / (n.x\*n.x + n.y\*n.y + n.z\*n.z);

pp.y = (n.y\*n.z\*A.z + n.z\*n.z\*p.y - n.y\*n.z\*p.z + n.y\*n.x\*A.x + n.x\*n.x\*p.y - \

n.x\*n.y\*p.x + n.y\*n.y\*A.y) / (n.x\*n.x + n.y\*n.y + n.z\*n.z);

pp.z = (n.x\*A.x\*n.z + n.x\*n.x\*p.z - n.x\*p.x\*n.z + n.y\*A.y\*n.z + n.y\*n.y\*p.z - \

n.y\*p.y\*n.z + n.z\*n.z\*A.z) / (n.x\*n.x + n.y\*n.y + n.z\*n.z);

return pp;

}