计算几何

*Computational Geometry*

*South China University of Technology*

*Author: 916852*

**一：基本公式**

**1.1 三角形........................................................... 6**

**1.2 四边形........................................................... 6**

**1.3 正n边形......................................................... 6**

**1.4 圆............................................................... 6**

**1.5 棱柱............................................................. 7**

**1.6 棱锥............................................................. 7**

**1.7 棱台............................................................. 7**

**1.8 圆柱............................................................. 7**

**1.9 圆锥............................................................. 7**

**1.10 圆台............................................................ 7**

**1.11 球.............................................................. 7**

**1.12 球台............................................................ 7**

**1.13 球扇形.......................................................... 8**

**二：点、线**

**2.1 结构定义........................................................... 8**

**2.2 向量p绕着圆点转动radian（弧度） 返回得到的点........................ 9**

**2.3 二维叉乘 返回a × b................................................. 9**

**2.4 二维点乘 返回a • b................................................. 9**

**2.5 二维两点距离....................................................... 9**

**2.6 向量a, b夹角的余弦值(弧度制) ...................................... 9**

**2.7 向量a, b夹角的正弦值(弧度制)....................................... 9**

**2.8 判断a,b,c三点共线................................................. 9**

**2.9 判断点在线段的位置................................................. 9**

**2.10 判断线段ab和cd是否相交.........................................10**

**2.11 判断直线ab和线段cd是否相交.....................................10**

**2.12 判断直线ab和直线cd是否相交.....................................11**

**2.13 判断两点在线段的同侧或异侧.........................................11**

**2.14 求线段所在直线的一般方程..........................................11**

**2.15 求点关于直线的对称点.............................................12**

**2.16 点到直线的最近距离...............................................12**

**2.17 点到线段的最近距离...............................................12**

**2.18 两线段最近距离...................................................12**

**2.19 向量夹角........................................................12**

**2.20 动点共线方程....................................................12**

**2.21 最近点对........................................................13**

**三：三角形**

**3.1 结构定义 ........................................................14**

**3.2 点在三角形内判定..................................................14**

**3.3 三角形覆盖k次面积并（可扩展为任意多边形，二叉空间划分）.............14**

**3.4 三角形四心........................................................16**

**3.5 三角形费马点......................................................17**

**四：圆**

**4.1 结构定义.........................................................17**

**4.2 点与圆的切点.....................................................18**

**4.3 圆的公切线........................................................18**

**4.4 线段与圆交点......................................................19**

**4.5 圆与圆交点........................................................19**

**4.6 圆的面积并........................................................20**

**4.7 圆覆盖k次面积并..................................................20**

**4.8 圆与多边形面积交...................................................21**

**4.9 点集最小圆覆盖.....................................................23**

**五：凸包多边形**

**5.1 andrew求凸包.....................................................23**

**5.2 点在多边形内判定...................................................24**

**5.3 旋转卡壳求凸包直径.................................................24**

**5.4 旋转卡壳求凸包上最大三角形面积......................................24**

**5.5 旋转卡壳求凸包最近距离.............................................25**

**5.6 logn直线切割凸包.................................................26**

**5.7 动态凸包..........................................................28**

**5.8 任意多边形最大内切圆(点+线=3限制内切圆)............................30**

**5.9 多边形面积并......................................................33**

**六：概率算法**

**6.1 最小球覆盖........................................................34**

**6.2 二维费马点........................................................34**

**七：平面问题**

**7.1 半平面交..........................................................36**

**7.2 PSLG平面直线图...................................................37**

**八：三维几何**

**8.1 三维叉乘..........................................................42**

**8.2 三维旋转矩阵......................................................43**

**8.3 三维旋转模型......................................................43**

**8.4 三维凸包相关......................................................44**

**8.5 三维光线反射......................................................48**

**8.6 点到直线距离......................................................48**

**8.7 点到线段距离......................................................49**

**8.8 两直线距离........................................................49**

**8.9 两线段距离........................................................49**

**8.10 直线相交判定.....................................................50**

**8.11 线段相交判定.....................................................50**

**8.12 点关于直线的对称点................................................50**

**8.13 点到平面距离.....................................................51**

**8.14 点在平面投影.....................................................51**

**8.15 点关于平面的对称点................................................51**

**8.16 直线与平面交点...................................................51**

**8.17 线段与平面交点...................................................51**

**8.18 直线与平面位置关系判定............................................51**

**8.19 两平面位置关系判定................................................52**

**8.20 平面交线.........................................................52**

**8.21 平面距离.........................................................52**

**8.22 点在空间三角形内判定.............................................52**

**8.23 线段和空间三角形的位置关系........................................53**

**8.24 经纬度坐标转笛卡尔坐标............................................53**

**8.25 球面距离........................................................53**

**九：数据结构优化算法**

**9.1 K-D树.........................................................53**

**十：其他**

**10.1 欧拉四面体公式...................................................55**

**10.2 simpson数值积分................................................55**

**10.3 常用积分公式.....................................................56**

**10.4 三角函数.........................................................56**

**一：基本公式**

* 1. **三角形**

**半周长 P = ( a + b + c ) / 2**

**面积 S = absin(C) / 2 = sqrt( P (P – a)(P - b)(P - c) )**

**余弦定理 2bc\*cos(A) = b^2 + c^2 – a^2**

**中线 Ma = sqrt( 2 ( b^2 + c^2 ) – a^2 ) / 2**

**= sqrt( b^2 + c^2 + 2bc\*cos(A) ) / 2**

**角平分线 Ta = sqrt( bc ( (b + c)^2 - a^2) ) / (b + c)**

**= 2bc\*cos(A/2) / (b + c)**

**高线 Ha = bsin(C) = csin(B)**

**= sqrt( b^2 - (( a^2 + b^2 - c^2 ) / (2a)) ^ 2 )**

**内切圆半径 r = S/P = a\*sin(B/2) \* sin(C/2) / sin((B+C)/2)**

**= 4R\*sin(A/2) \* sin(B/2) \* sin(C/2)**

**= sqrt( (P – a) \* (P – b) \* (P – c) / P )**

**= P \* tan(A/2) \* tan(B/2) \* tan(C/2)**

**外接圆半径 R = abc / (4S) = a / (2sin(A))**

**= b / (2sin(B)) = c / (2sin(C))**

**1.2 四边形**

**D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角**

**1. a^2 + b^2 + c^2 + d^2 = D1^2 + D2^2 + 4M^2**

**2. S = D1D2sin(A)/2**

**(以下对圆的内接四边形)**

**3. ac + bd = D1 \* D2**

**4. S = sqrt((P - a)(P - b)(P - c)(P - d)), P 为半周长**

**1.3 正 n 边形**

**R 为外接圆半径,r 为内切圆半径**

**1. 中心角 A = 2 \* PI / n**

**2. 内角 C = (n - 2) \* PI / n**

**3. 边长 a = 2sqrt(R^2 - r^2) = 2R \* sin(A/2) = 2r \* tan(A/2)**

**4. 面积 S = nar / 2 = nr^2 \* tan(A/2)**

**= nR^2 \* sin(A) / 2 = na^2 / (4tan(A/2))**

**1.4 圆**

**r为半径，A为角度**

**弧长 l = rA**

**弦长 a = 2 \* sqrt(2hr - h^2) = 2rsin(A/2)**

**弓形高 h = r - sqrt(r^2 - a^2/4) = r(1 - cos(A/2)) = atan(A/4) / 2**

**扇形面积 S1 = r \* l / 2 = r^2 \* A / 2**

**弓形面积 S2 = (rl - a(r - h)) / 2 = r^2 \* (A - sin(A)) / 2**

**1.5 棱柱**

**1. 体积 V = Ah, A 为底面积,h 为高**

**2. 侧面积 S = lp, l 为棱长, p 为直截面周长**

**3. 全面积 T=S+2A**

**1.6 棱锥**

**1. 体积 V = Ah/3, A 为底面积,h 为高**

**(以下对正棱锥)**

**2. 侧面积 S = lp/2, l 为斜高,p 为底面周长**

**3. 全面积 T = S + A**

**1.7 棱台**

**1. 体积 V = ( A1 + A2 + sqrt(A1A2) ) \* h / 3, A1.A2 为上下底面积,h 为高**

**(以下为正棱台)**

**2. 侧面积 S = (p1 + p2) l / 2, p1.p2 为上下底面周长,l 为斜高**

**3. 全面积 T = S + A1 + A2**

**1.8 圆柱**

**1. 侧面积 S = 2 \* PI \* r \* h**

**2. 全面积 T = 2 \* PI \* r \* (h + r)**

**3. 体积 V = PI \* r^2 \* h**

**1.9 圆锥**

**母线 l = sqrt(h^2 + r^2)**

**侧面积 S = PI \* r \* l**

**全面积 T = PI \* r ( l + r )**

**体积 V = PI \* r^2 \* h / 3**

**1.10 圆台**

**母线 l = sqrt( h^2 + (r1 - r2)^2 )**

**侧面积 S = PI \* ( r1 + r2 ) \* l**

**全面积 T = PI \* r1 \* (l + r1) + PI \* r2 \* (l + r2)**

**体积 V = PI \* (r1^2 + r2^2 + r1r2) \* h / 3**

**1.11 球**

**1. 全面积 T = 4 \* PI \* r^2**

**2. 体积 V = 4 \* PI \* r^3 / 3**

**1.12 球台**

**1. 侧面积 S = 2 \* PI \* r \* h**

**2. 全面积 T = PI \* (2rh + r1^2 + r2^2)**

**3. 体积 V = PI \* h(3 \* (r1^2 + r2^2) + h^2) / 6**

**1.13 球扇形**

**1. 全面积 T = PI \* r(2h + r0), h 为球冠高,r0 为球冠底面半径**

**2. 体积 V = 2 \* PI \* r^2 \* h / 3**

**3. 球缺 V = V球扇 - V圆锥 = PI \* h \* h \* (3r - h) / 3**

**二：点、线**

**2.1结构定义**

**#define MP make\_pair**

**#define LL long long**

**#define uLL unsigned long long**

**const double PI = acos(-1.0);**

**const double eps = 1e-8;**

**struct point {**

**double x, y, z;**

**point(){}**

**point( double x, double y, double z ) : x(x), y(y), z(z) {}**

**point operator - ( const point b ) const {**

**return point( x - b.x, y - b.y, z - b.z );**

**}**

**point operator + ( const point b ) const {**

**return point( x + b.x, y + b.y, z + b.z );**

**}**

**point operator \* ( double d ) const {**

**return point( x \* d, y \* d, z \* d );**

**}**

**point operator / ( double d ) const {**

**return point( x / d, y / d, z / d );**

**}**

**double len() {**

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

**}**

**void input() {**

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

**}**

**};**

**struct Line { point a, b; };**

**struct Nline { int a, b, c; }; // ax + by + c = 0 一般方程**

**struct Circle { double r; point c; };**

**struct Sphere { double r; point3 c; }; //球体**

**int dcmp( double x ){  
 return (x > eps) – (x < -eps) ;**

**}**

**2.2向量p绕着圆点转动radian（弧度） 返回得到的点**

**point rotate(point p, double radian) {**

**double c = cos(radian), s = sin(radian);**

**point res;**

**res.x = p.x \* c - p.y \* s;**

**res.y = p.y \* c + p.x \* s;**

**return res;**

**}**

**2.3二维叉乘 返回a × b**

**double cross( point a, point b ) {**

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

**}**

**2.4二维点乘 返回 a · b**

**double dot( point a, point b ) {**

**return a.x \* b.x + a.y \* b.y;**

**}**

**2.5二维两点距离**

**double dis(point a, point b) {**

**return sqrt(dot(a – b, a - b));**

**}**

**2.6 向量a, b夹角的余弦值(弧度制)**

**double cos(point a, point b) {**

**return dot(a, b) / a.len() / b.len();**

**}**

**2.7 向量a, b夹角的正弦值(弧度制)**

**double sin(point a, point b) {**

**return fabs( cross(a, b) / a.len() / b.len() );**

**}**

**2.8判断a,b,c三点共线**

**bool in\_line(point a, point b, point b) {**

**return dcmp( cross(b - a, c - a) ) == 0 ;**

**}**

**2.9 判断点在线段的位置**

**前提假设a、b、x共线**

**返回：**

**x在seg(a,b)内：-1**

**x在seg(a,b)上：0**

**x在seg(a,b)外：1**

**int btw(point x, point a, point b) {**

**return dcmp( dot(a - x, b - x) );**

**}**

**2.10判断线段ab和cd是否相交**

**类型 返回 res**

**--------------------------------------**

**1. 不相交 0 不变**

**2. 规范相交 1 交点 （交叉）**

**3. 非规范相交 2 不变 （端点在另一线段，有重叠段）**

**int segCross(point a, point b, point c, point d, point &res) {**

**double s1, s2;**

**int d1, d2, d3, d4;**

**d1 = dcmp( s1 = cross(b - a, c - a) );**

**d2 = dcmp( s2 = cross(b - a, d - a) );**

**d3 = dcmp( cross(d - c, a - c) );**

**d4 = dcmp( cross(d - c, b - c) );**

**if( (d1^d2) == -2 && (d3^d4) == -2 ){**

**res.x = (c.x \* s2 - d.x \* s1) / (s2 - s1);**

**res.y = (c.y \* s2 - d.y \* s1) / (s2 - s1);**

**return 1;**

**}**

**if( d1 == 0 && btw(c, a, b) <= 0 ||**

**d2 == 0 && btw(d, a, b) <= 0 ||**

**d3 == 0 && btw(a, c, d) <= 0 ||**

**d4 == 0 && btw(b, c, d) <= 0)**

**return 2;**

**return 0;**

**}**

**2.11 判断直线ab和线段cd是否相交**

**类型 返回 res**

**--------------------------------------**

**1. 不相交 0 不变**

**2. 规范相交 1 交点 （交叉）**

**3. 非规范相交 2 不变 （线段端点在直线，有重叠段）**

**int segLineCross(point a, point b, point c, point d, point &res) {**

**double s1, s2;**

**int d1, d2;**

**d1 = dcmp( s1 = cross(b - a, c - a) );**

**d2 = dcmp( s2 = cross(b - a, d - a) );**

**if( (d1^d2) == -2 ) {**

**res.x = (c.x \* s2 - d.x \* s1) / (s2 - s1);**

**res.y = (c.y \* s2 - d.y \* s1) / (s2 - s1);**

**return 1;**

**}**

**if( d1 == 0 || d2 == 0 ) return 2;**

**return 0;**

**}**

**2.12 判断直线ab和直线cd是否相交**

**类型 返回 res**

**--------------------------------------**

**1. 不相交（平行） 0 不变**

**2. 规范相交 1 交点**

**3. 非规范相交（重合） 2 不变**

**int lineCross(point a, point b, point c, point d, point &res) {**

**double s1, s2;**

**s1 = cross(b - a, c - a);**

**s2 = cross(b - a, d - a);**

**if( dcmp(s1) == 0 && dcmp(s2) == 0 ) return 2;**

**if( dcmp(s2 - s1) == 0 ) return 0;**

**res.x = (c.x \* s2 - d.x \* s1) / (s2 - s1);**

**res.y = (c.y \* s2 - d.y \* s1) / (s2 - s1);**

**return 1;**

**}**

**2.13 判断两点在线段的同侧或异侧...............**

**类型 返回**

**----------------------------**

**1. 某点在线段上 0**

**2. 同侧 1**

**3. 异侧 -1**

**int pointside( point a, point b, Line l ) {**

**return dcmp( cross(a – l.a, l.b – l.a) \* cross(b – l.a, l.b – l.a) );**

**}**

**2.14 求线段所在直线一般方程**

**Nline lfs(point p1, point p2) //line from segment**

**{**

**Nline tmp;**

**tmp.a = p2.y - p1.y;**

**tmp.b = p1.x - p2.x;**

**tmp.c = p2.x \* p1.y - p1.x \* p2.y;**

**return tmp;**

**}**

**2.15 求点关于直线的对称点**

**point spl(point p, Nline L) { // symmetrical point of Line**

**point p2;**

**double d;**

**d = L.a \* L.a + L.b \* L.b;**

**p2.x = (L.b \* L.b \* p.x - L.a \* L.a \* p.x -**

**2 \* L.a \* L.b \* p.y - 2 \* L.a \* L.c) / d;**

**p2.y = (L.a \* L.a \* p.y - L.b \* L.b \* p.y -**

**2 \* L.a \* L.b \* p.x - 2 \* L.b \* L.c) / d;**

**return p2;**

**}**

**2.16 点到直线的最近距离**

**double ptoline( point p, point a, point b ){**

**return fabs(cross(p - a, b - a)) / dis(a, b);**

**}**

**double ptoline( point p, Nline l ){**

**return ( p.x \* l.a + p.y \* l.b + l.c ) / sqrt( l.a \* l.a + l.b \* l.b );**

**}**

**2.17 点到线段的最近距离**

**double ptoseg( point p, point a, point b ) {**

**if( dcmp( dot(p - a, b - a) ) <= 0 ) return dis(p, a);**

**if( dcmp( dot(p - b, a - b) ) <= 0 ) return dis(p, b);**

**return fabs(cross(p - a, b - a)) / dis(a, b);**

**}**

**2.18 两线段最近距离**

**相交距离为0，否则枚举两条线段的端点到另一线段的距离**

**2.19 向量夹角**

**double angle( point a, point b ) {**

**double k = dot(a, b) / a.len() / b.len();**

**k = max(k, -1.0); k = min(k, 1.0);**

**return acos( k );**

**}**

**2.20 动点共线方程**

**//动点point( pnt[i].x + dx[i] \* t, pnt[i].y + dy[i] \* t ) 求大于0的解**

**void cal( int &i, int &j, int &k, double &t1, double &t2 ) {**

**double a1 = pnt[i].x - pnt[j].x, b1 = dx[i] - dx[j];**

**double a2 = pnt[k].y - pnt[j].y, b2 = dy[k] - dy[j];**

**double a3 = pnt[i].y - pnt[j].y, b3 = dy[i] - dy[j];**

**double a4 = pnt[k].x - pnt[j].x, b4 = dx[k] - dx[j];**

**double a = b1 \* b2 - b3 \* b4;**

**double b = a1 \* b2 + a2 \* b1 - a3 \* b4 - a4 \* b3;**

**double c = a1 \* a2 - a3 \* a4;**

**double dlt = b \* b - 4.0 \* a \* c;**

**if( dcmp(a) == 0 ) {**

**if( dcmp(b) == 0 ) { //c == 0 无穷解 c != 0 无解**

**t1 = t2 = -1.0;**

**}**

**else t1 = -c / b, t2 = -1.0;**

**}**

**else if( dcmp(dlt) == 0 ) t1 = -b / (2 \* a), t2 = -1.0;**

**else if( dlt > 0 ) {**

**t1 = (-b - sqrt(dlt)) / (2 \* a);**

**t2 = (-b + sqrt(dlt)) / (2 \* a);**

**}**

**else t1 = t2 = -1.0;**

**}**

**2.21 最近点对**

**//调用closep(p, 0, n)**

**point p[mxn], py[mxn];**

**bool cmpy( point a, point b ) {**

**return a.y < b.y;**

**}**

**double closep( point \*p, int ll, int rr ) {**

**if( rr - ll <= 1 ) return inf; //周长最小三角形返回MP(inf,-1)**

**int m = (ll + rr) >> 1;**

**double midx = p[m].x;**

**double res = min( closep(p, ll, m), closep(p, m, rr) );**

**inplace\_merge(p + ll, p + m, p + rr, cmpy);**

**double x1 = midx - res, x2 = midx + res;**

**int len = 0;**

**for( int i = ll; i < rr; ++i )**

**if( p[i].x > x1 && p[i].x < x2 )**

**py[len++] = p[i];**

**for( int i = 0; i < len; ++i )**

**for( int j = i + 1; j < len && py[j].y < py[i].y + res; ++j )**

**res = min(res, dis(py[i], py[j]));**

**//可以再加一层for(k = j + 1 -> len && py[k].y < py[i].y + res) 求周长最小的三角形 res初始化为当前最优三角形周长一半 函数返回pair<周长，id>**

**return res;**

**}**

**三：三角形**

**3.1. 结构定义**

**struct triangle {**

**point a, b, c;**

**void input() {**

**a.input(); b.input(); c.input();**

**if( dcmp(cross(b - a, c - a)) < 0 ) //保障逆时针序**

**swap(b, c);**

**}**

**};**

**3.2 点在三角形内判定**

**//判断点o是否在△abc内**

**bool intrian( point o, point a, point b, point c ) {**

**if( dcmp(cross(b - a, o - a)) < 0 ) return false;**

**if( dcmp(cross(c - b, o - b)) < 0 ) return false;**

**if( dcmp(cross(a - c, o - c)) < 0 ) return false;**

**return true;**

**}**

**3.3 三角形覆盖k次面积并（可扩展为任意多边形，二叉空间划分）**

**//初始无穷大平面 递归切割**

**struct polygon {**

**int n;**

**vector<point> p;**

**double area() {**

**double s = 0;**

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

**s += cross(p[i-1] - p[0], p[i] - p[0]);**

**return fabs(s) / 2;**

**}**

**point center() { //多边形重心**

**double s = 0, sx = 0, sy = 0;**

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

**double x = p[0].x + p[i-1].x + p[i].x;**

**double y = p[0].y + p[i-1].y + p[i].y;**

**double tmps = cross(p[i-1] - p[0], p[i] - p[0]) / 2;**

**s += tmps;**

**sx += x \* tmps;**

**sy += y \* tmps;**

**}**

**return point( sx / s / 3, sy / s / 3 );**

**}**

**}g[mxn]; Line L[mxn]; triangle sjx[mxn]; int cnt;**

**void add( point a, point b, int id, int cur ) {**

**g[cur].p.clear();**

**for( int i = 0; i < g[id].n; ++i ) {**

**int d1 = dcmp( cross(b - a, g[id].p[i] - a) );**

**int d2 = dcmp( cross(b - a, g[id].p[i+1] - a) );**

**if( d1 >= 0 ) g[cur].p.push\_back( g[id].p[i] );**

**if( (d1 ^ d2) == -2 ) {**

**point x = linecross( a, b, g[id].p[i], g[id].p[i+1] );**

**g[cur].p.push\_back(x);**

**}**

**}**

**g[cur].n = g[cur].p.size();**

**}**

**void dfs( int dep, int id ) {**

**g[id].p.push\_back(g[id].p[0]);**

**if( dep == m ) return ;**

**point a = L[dep].s, b = L[dep].t;**

**bool nolft = true, norht = true;**

**for( int i = 0; i < g[id].n; ++i ) {**

**int d = dcmp( cross(b - a, g[id].p[i] - a) );**

**if( d > 0 ) nolft = false;**

**if( d < 0 ) norht = false;**

**}**

**if( nolft || norht ) {**

**dfs( dep + 1, id );**

**return ;**

**}**

**bad[id] = true;**

**++cnt;**

**add( a, b, id, cnt );**

**dfs( dep + 1, cnt );**

**++cnt;**

**add( b, a, id, cnt );**

**dfs( dep + 1, cnt );**

**}**

**int t, n;**

**scanf( "%d", &t );**

**while( t-- ) {**

**scanf( "%d", &n );**

**m = cnt = 0;**

**memset( ans, 0, sizeof(ans) );**

**memset( bad, 0, sizeof(bad) );**

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

**sjx[i].input();**

**L[m++] = Line( sjx[i].a, sjx[i].b );**

**L[m++] = Line( sjx[i].b, sjx[i].c );**

**L[m++] = Line( sjx[i].c, sjx[i].a );**

**}**

**g[0].p.clear();**

**g[0].p.push\_back( point(-200, -200) ); //确保足够大**

**g[0].p.push\_back( point(200, -200) );**

**g[0].p.push\_back( point(200, 200) );**

**g[0].p.push\_back( point(-200, 200) );**

**g[0].n = g[0].p.size();**

**dfs(0, 0);**

**for( int i = 0; i <= cnt; ++i ) {**

**if( bad[i] ) continue;**

**point x = g[i].center();**

**int num = 0;**

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

**if( intrian(x, sjx[j].a, sjx[j].b, sjx[j].c) )**

**++num;**

**ans[num] += g[i].area();**

**}**

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

**printf( "%.10lf\n", ans[i] );**

**}**

**3.4 三角形四心**

**重心：中线交点，近边三等分点，三角形内到三边距离之积最大，到三顶点距离的平方和最小**

**point zhong(point a, point b, point c) {**

**return (a + b + c) / 3;  
}**

**外心：三条垂直平分线交点，外接圆圆心**

**point wai( point& a, point& b, point& c ) {**

**point res;**

**double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 \* a1 + b1 \* b1)/2;**

**double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 \* a2 + b2 \* b2)/2;**

**double d = a1 \* b2 - a2 \* b1;**

**res.x = a.x + (c1 \* b2 - c2 \* b1) / d;**

**res.y = a.y + (a1 \* c2 - a2 \* c1) / d;**

**return res;**

**}**

**内心：三内角平分线交点，内接圆圆心**

**point nei(point a, point b, point c) {**

**double A = dis(b, c), B = dis(a, c), C = dis(a, b);**

**double P = A + B + C;**

**return a \* (A/P) + b \* (B/P) + c \* (C/P);  
}**

**旁心：一内角平分线和另外两角的外角平分线（好像木有啥用）**

**3.5 三角形费马点**

**费马点：到所有点距离之和最小的点**

**有△ABC，设∠A大于120度，则点A为费马点**

**否则，费马点到三点的连线等分费马点周角，故此类三角形费马点也是三角形等角中心**

**从三角形三边向外做等边三角形A’BC, AB’C, ABC’，则AA’,BB’,CC’三线共点于费马点**

**//凸四边形费马点：对角线交点**

**//凹四边形费马点：凹点**

**//Find three numbers r + s + t = 1, which make p = r \* a + s \* b + t \* c**

**void parametric(point p, point a, point b, point c) {**

**double d = cross(b – a, c - a);**

**r = cross(b – p, c - p) / d;**

**s = cross(p – a, c - a) / d;**

**t = cross(b – a, p - a) / d;**

**}**

**四：圆**

**4.1 结构定义**

**struct circle {**

**point c;**

**double r;**

**int id;**

**circle(){}**

**circle(point c, double r) : c(c), r(r) {}**

**point getp(double ang) { //圆上相对圆心以ang为极角的点**

**return point(c.x + r \* cos(ang), c.y + r \* sin(ang), id);**

**}**

**void input(int k) {**

**id = k;**

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

**}**

**}**

**4.2 点与圆的切点**

**//前提点在圆外**

**int ptancircle( point k, circle a ) {**

**point u = k - a.c;**

**double len = u.len();**

**double ang = acos( a.r / len );**

**double bas = atan2( u.y, u.x );**

**pnt[num++] = a.getp( bas + ang );**

**pnt[num++] = a.getp( bas - ang );**

**return 2;**

**}**

**4.3 圆的公切线**

**//精度曾卡1e-15， pnt保存所有切点， 可分别保存在另外两个数组**

**int getTangents( circle a, circle b ) {**

**int cnt = 0;**

**if( a.r < b.r ) swap(a, b);**

**double d2 = dis2(a.c, b.c);**

**double rcha = a.r - b.r;**

**double rsum = a.r + b.r;**

**if( dcmp(d2 - rcha \* rcha) < 0 ) return 0;**

**double bas = atan2(b.c.y - a.c.y, b.c.x - a.c.x);**

**if( dcmp(d2) == 0 && dcmp(a.r - b.r) == 0 ) return -1;**

**if( dcmp(d2 - rcha \* rcha) == 0 ) {**

**pnt[num++] = a.getp(bas);**

**pnt[num++] = b.getp(bas);**

**cnt++;**

**return 1;**

**}**

**double ang = acos( (a.r - b.r) / sqrt(d2) );**

**pnt[num++] = a.getp(bas + ang); pnt[num++] = a.getp(bas - ang); cnt++;**

**pnt[num++] = b.getp(bas + ang); pnt[num++] = b.getp(bas - ang); cnt++;**

**if( dcmp(d2 - rsum \* rsum) == 0 ) {**

**pnt[num++] = a.getp(bas); pnt[num++] = b.getp(bas + pi);**

**cnt++;**

**}**

**else if( dcmp(d2 - rsum \* rsum) > 0 ) {**

**double ang = acos( (a.r + b.r) / sqrt(d2) );**

**pnt[num++] = a.getp(bas + ang); pnt[num++] = a.getp(bas - ang); cnt++;**

**pnt[num++] = b.getp(pi + bas + ang); pnt[num++] = b.getp(pi + bas - ang); cnt++;**

**}**

**return cnt;**

**}**

**4.4 线段与圆交点**

**//圆心c，半径r，线段ab，交点为res1，res2，返回k是交点个数**

**//#define sqr(x) ((x)\*(x))**

**int seg\_cir(point c, double r, point a, point b, point &res1, point &res2) {**

**int k = 0;**

**double aa = sqr(a.x - b.x) + sqr(a.y - b.y);**

**double bb = 2 \* ((b.x - a.x)\*(a.x - c.x) + (b.y - a.y)\*(a.y - c.y));**

**double cc = sqr(c.x) + sqr(c.y) + sqr(a.x) + sqr(a.y) - r \* r - 2 \* (c.x \* a.x + c.y \* a.y);**

**if( dcmp( bb \* bb - 4 \* aa \* cc ) >= 0 ) {**

**double u1 = (-bb + sqrt(bb \* bb - 4 \* aa \* cc)) / 2.0 / aa;**

**double u2 = (-bb - sqrt(bb \* bb - 4 \* aa \* cc)) / 2.0 / aa;**

**if( u1 > u2 && dcmp(u2) >= 0 ) swap(u1, u2);**

**if( dcmp(u1) >= 0 && dcmp(u1-1) <= 0 ) {**

**res1.x = a.x + u1 \* (b.x - a.x);**

**res1.y = a.y + u1 \* (b.y - a.y);**

**//if( dcmp(res1.y - c.y) <= 0 ) res1.ok = true; 下半圆判定**

**++k;**

**}**

**if( dcmp(u1-u2) && dcmp(u2) >= 0 && dcmp(u2-1) <= 0 ) {**

**res2.x = a.x + u2 \* (b.x - a.x);**

**res2.y = a.y + u2 \* (b.y - a.y);**

**//if( dcmp(res2.y - c.y) <= 0 ) res2.ok = true; 下半圆判定**

**++k;**

**}**

**}**

**return k;**

**}**

**4.5 圆与圆交点**

**//两圆圆心为c1,c2，半径为r1,r2，交点保存在k1,k2**

**//两圆重合需要自行特判,返回交点个数**

**int CirCrossCir(point c1, double r1, point c2, double r2, point &k1, point &k2)**

**{**

**double mx = c2.x - c1.x, sx = c2.x + c1.x, mx2 = mx \* mx;**

**double my = c2.y - c1.y, sy = c2.y + c1.y, my2 = my \* my;**

**double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) \* (sq - sqr(r1 + r2));**

**if (dcmp(d) < 0) return 0;**

**if (dcmp(d) == 0) d = 0; else d = sqrt(d);**

**double x = mx \* ((r1 + r2) \* (r1 - r2) + mx \* sx) + sx \* my2;**

**double y = my \* ((r1 + r2) \* (r1 - r2) + my \* sy) + sy \* mx2;**

**double dx = mx \* d, dy = my \* d; sq \*= 2;**

**k1.x = (x - dy) / sq; k1.y = (y + dx) / sq;**

**k2.x = (x + dy) / sq; k2.y = (y - dx) / sq;**

**if (d > eps) return 2;**

**else return 1;**

**}**

**4.6 圆的面积并**

**把下面那个覆盖k次加起来，都是O(n^2)的复杂度**

**4.7 圆覆盖k次面积并**

**//area[i]保存覆盖i次的面积**

**#define sqr(x) ((x)\*(x))**

**struct Circle {**

**point c;**

**double r, ang;**

**int d;**

**Circle(){}**

**Circle(point c, double ang = 0, int d = 0):c(c), ang(ang), d(d) {}**

**void input() {**

**c.input(); d = 1;**

**scanf( "%lf", &r );**

**}**

**}cir[mxn], tp[mxn \* 2];**

**bool circmp(const Circle& a, const Circle& b) {**

**return dcmp(a.r - b.r) < 0;**

**}**

**bool cmp(const Circle& a, const Circle& b) {**

**if( dcmp(a.ang - b.ang) )**

**return a.ang < b.ang;**

**return a.d > b.d;**

**}**

**double calc(Circle o, Circle a, Circle b) {**

**double ans = (b.ang - a.ang) \* sqr(o.r)**

**- cross(a.c - o.c, b.c - o.c) + cross(a.c - point(0,0), b.c - point(0,0));**

**return ans / 2;**

**}**

**void CirUnion(Circle cir[], int n) {**

**Circle res1, res2;**

**sort( cir, cir + n, circmp );**

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

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

**if( dcmp(dis(cir[i].c, cir[j].c) + cir[i].r - cir[j].r) <= 0 )**

**cir[i].d++;**

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

**int tn = 0, cnt = 0;**

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

**if( i == j ) continue;**

**if( CirCrossCir(cir[i].c, cir[i].r, cir[j].c, cir[j].r,**

**res2.c, res1.c) < 2) continue; //res2和res1不能交换**

**res1.ang = atan2(res1.c.y - cir[i].c.y, res1.c.x - cir[i].c.x);**

**res2.ang = atan2(res2.c.y - cir[i].c.y, res2.c.x - cir[i].c.x);**

**res1.d = 1; tp[tn++] = res1;**

**res2.d = -1; tp[tn++] = res2;**

**if( dcmp(res1.ang - res2.ang) > 0 ) cnt++;**

**}**

**tp[tn++] = Circle(point(cir[i].c.x - cir[i].r, cir[i].c.y), pi, -cnt);**

**tp[tn++] = Circle(point(cir[i].c.x - cir[i].r, cir[i].c.y), -pi, cnt);**

**sort( tp, tp + tn, cmp );**

**int p, s = cir[i].d + tp[0].d;**

**for( int j = 1; j < tn; ++j ) {**

**p = s; s += tp[j].d;**

**area[p] += calc( cir[i], tp[j - 1], tp[j] );**

**}**

**}**

**}**

**void solve() {**

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

**cir[i].input();**

**memset(area, 0, sizeof(area));**

**CirUnion(cir, n);**

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

**area[i] -= area[i + 1];**

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

**printf("[%d] = %.3lf\n", i, area[i]);**

**}**

**4.8 圆与多边形面积交**

**//圆的圆心固定为(0,0)，如果不是进行坐标变换，半径是R**

**double R;**

**point point::change() { //加到点结构体的函数**

**return point( R \* x / d, R \* y / d );**

**}**

**double calang(point a, point b) { //有方向的极角差，不同于向量夹角**

**double t = atan2(a.y, a.x) - atan2(b.y, b.x);**

**while( dcmp(t - pi) > 0 ) t -= pi\*2;**

**while( dcmp(t + pi) < 0 ) t += pi\*2;**

**return t;**

**}**

**double solve( int n, point \*p ) {**

**double ans = 0, ang = 0;**

**point res1, res2, o(0, 0);**

**p[n] = p[0]; //点加一个变量d保存点到原点的距离**

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

**p[i].d = p[i].len();**

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

**if( dcmp(p[i-1].d - R) < 0 ) {**

**if( dcmp(p[i].d - R) < 0 )**

**ans += cross(p[i-1], p[i]);**

**else {**

**seg\_cir(o, R, p[i-1], p[i], res1, res2); //线段与圆交**

**ans += cross(p[i-1], res1);**

**ang += calang(p[i].change(), res1);**

**}**

**}**

**else {**

**if( dcmp(p[i].d - R) < 0 ) {**

**seg\_cir(o, R, p[i-1], p[i], res1, res2);**

**ans += cross(res1, p[i]);**

**ang += calang(res1, p[i-1].change());**

**}**

**else {**

**if( seg\_cir(o, R, p[i-1], p[i], res1, res2) == 2 ) {**

**ang += calang(res1, p[i-1].change());**

**ans += cross(res1, res2);**

**ang += calang(p[i].change(), res2);**

**}**

**else**

**ang += calang(p[i].change(), p[i-1].change());**

**}**

**}**

**}**

**ans = ans / 2 + ang \* R \* R / 2;**

**return fabs(ans);**

**}**

**4.9 点集最小圆覆盖**

**//期望复杂度是线性的**

**void minCirle( int n, point \*p, point &c, double &r ) {**

**random\_shuffle( p, p + n );**

**c = p[0]; r = 0;**

**for( int i = 1; i < n; ++i ) if( dcmp(dis(p[i], c) - r) > 0 ) {**

**c = p[i]; r = 0;**

**for( int j = 0; j < i; ++j ) if( dcmp(dis(p[j], c) - r) > 0 ) {**

**c.x = 0.5 \* (p[i].x + p[j].x);**

**c.y = 0.5 \* (p[i].y + p[j].y);**

**r = dis(p[j], c);**

**for( int k = 0; k < j; ++k ) if( dcmp(dis(p[k], c) - r) > 0 ) {**

**c = wai(p[i], p[j], p[k]); //三角形外心**

**r = dis(p[i], c);**

**}**

**}**

**}**

**}**

**五：凸包多边形**

**5.1 andrew求凸包**

**//点按x坐标从小到大排序，相同按y排序, double要加dcmp,PS先按y排序也可以**

**//凸包边上无共线点，如果要保留共线点，去掉cross后面的等号**

**int andrew( int n ) {**

**sort( pnt, pnt + n );**

**int m = 0;**

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

**while( m > 1 && cross( res[m-1] - res[m-2], pnt[i] - res[m-1] ) <= 0 )**

**--m;**

**res[m++] = pnt[i];**

**}**

**int k = m;**

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

**while( m > k && cross( res[m-1] - res[m-2], pnt[i] - res[m-1] ) <= 0 )**

**--m;**

**res[m++] = pnt[i];**

**}**

**if( n > 1 ) --m;**

**return m;**

**}**

**5.2点在多边形内判定**

**//double要dcmp**

**bool ponseg( point p, point a, point b ) {**

**return cross( a - p, b - p ) == 0 && dot( a - p, b - p ) <= 0;**

**}**

**// 0:外, 1:内, 2:边**

**int pointInPolygon( point cp, point\* p, int n ) {**

**int w = 0;**

**p[n] = p[0];**

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

**if( ponseg(cp, p[i], p[i+1]) )**

**return 2;**

**int k = dcmp(cross(p[i+1] - p[i], cp - p[i]));**

**int d1 = dcmp(p[i].y - cp.y);**

**int d2 = dcmp(p[i+1].y - cp.y);**

**if( k > 0 && d1 <= 0 && d2 > 0 ) w++;**

**if( k < 0 && d2 <= 0 && d1 > 0 ) w--;**

**}**

**return w != 0;**

**}**

**5.3旋转卡壳求凸包直径**

**double maxd( point\* p, int n ) {**

**double ret = 0;**

**int j = 0;**

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

**while( (j + 1) % n != i && cross(p[(i+1)%n] - p[i], p[(j+1)%n] - p[i])**

**>= cross(p[(i+1)%n] - p[i], p[j] - p[i]) )**

**j = (j + 1) % n;**

**ret = max(ret, dis(p[i], p[j]));**

**ret = max(ret, dis(p[i+1], p[j]));**

**}**

**return ret;**

**}**

**5.4旋转卡壳求凸包上最大三角形面积**

**double maxarea( point\* p, int n ) {**

**int j = 1, k = 2;**

**LL ans = 0;**

**p[n] = p[0];**

**p[n+1] = p[1];**

**p[n+2] = p[2];**

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

**if( j == i ) j = (j+1)%n;**

**if( k == j ) k = (k+1)%n;**

**while( cross(p[j] - p[i], p[k] - p[i]) <**

**cross(p[j] - p[i], p[(k+1)%n] - p[i]) )**

**k = (k + 1) % n;**

**ans = max(ans, cross(p[j] - p[i], p[k] - p[i]));**

**while( cross(p[j] - p[i], p[k] - p[i]) <**

**cross(p[(j+1)%n] - p[i], p[k] - p[i]) )**

**j = (j + 1) % n;**

**ans = max(ans, cross(p[j] - p[i], p[k] - p[i]));**

**}**

**return ans / 2;**

**}**

**5.5旋转卡壳求凸包最近距离**

**double mind( point \*p, int np, point \*q, int nq ) {**

**int sp = 0, sq = 0;**

**for( int i = 1; i < np; ++i )**

**if( dcmp(p[i].y - p[sp].y) < 0 || dcmp(p[i].y - p[sp].y) == 0 && dcmp(p[i].x - p[sp].x) < 0 )**

**sp = i;**

**for( int i = 1; i < nq; ++i )**

**if( dcmp(q[i].y - q[sq].y) > 0 || dcmp(q[i].y - q[sq].y) == 0 && dcmp(q[i].x - q[sq].x) > 0 )**

**sq = i;**

**int tp = sp, tq = sq;**

**double ans = dis(p[sp], q[sq]);**

**do {**

**double len = cross(p[(sp+1)%np] - p[sp], q[sq] - q[(sq+1)%nq]);**

**if( dcmp(len) == 0 ) {**

**ans = min(ans, ptoseg(p[sp], q[sq], q[(sq+1)%nq]));**

**ans = min(ans, ptoseg(p[(sp+1)%np], q[sq], q[(sq+1)%nq]));**

**ans = min(ans, ptoseg(q[sq], p[sp], p[(sp+1)%np]));**

**ans = min(ans, ptoseg(q[(sq+1)%nq], p[sp], p[(sp+1)%np]));**

**sp = (sp + 1) % np; sq = (sq + 1) % nq;**

**}**

**else if( dcmp(len) > 0 ) {**

**ans = min(ans, ptoseg(q[sq], p[sp], p[(sp+1)%np]));**

**sp = (sp + 1) % np;**

**}**

**else {**

**ans = min(ans, ptoseg(p[sp], q[sq], q[(sq+1)%nq]));**

**sq = (sq + 1) % nq;**

**}**

**} while( tp != sp || tq != sq );**

**return ans;**

**}**

**5.6 logn直线切割凸包**

**//点结构重载小于号运算符 return ang < b.ang， res是凸包点集**

**//andrew排序务必先按y轴，保障凸包点集第一个点是y坐标最小，逆时针序**

**double cal\_ang( point& a, point& b ) {**

**double ang = atan2(b.y - a.y, b.x - a.x);**

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

**return ang;**

**}**

**double sum[mxn];**

**void init( point \*p, int n ) {**

**p[n] = p[0];**

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

**p[i].ang = cal\_ang(p[i], p[i+1]);**

**sum[0] = cross(p[0], p[1]);**

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

**sum[i] = sum[i-1] + cross(p[i], p[i+1]);**

**}**

**double get( int a, int b ) {**

**if( (--b) < 0 ) return 0;**

**if( (--a) < 0 ) return sum[b];**

**return sum[b] - sum[a];**

**}**

**int find( int beg, int maxlen, point s, point t, point \*p, int n ) {**

**int sign = dcmp(cross(t - s, p[beg] - s));**

**int l = 0, r = maxlen + 1;**

**while( r - l > 1 ) {**

**int m = (l + r) / 2;**

**if( dcmp(cross(t - s, p[(beg+m)%n] - s)) \* sign >= 0 )**

**l = m;**

**else**

**r = m;**

**}**

**return (beg + l) % n;**

**}**

**double line\_cut\_con( point s, point t, point \*p, int n ) {**

**double ang = cal\_ang(s, t), res;**

**point tmp, res1, res2;**

**p[n] = p[0];**

**tmp.ang = ang;**

**int a = upper\_bound(p, p + n, tmp) - p; a %= n;**

**tmp.ang = (ang + pi > 2 \* pi) ? ang - pi : ang + pi;**

**int b = upper\_bound(p, p + n, tmp) - p; b %= n;**

**int d1 = dcmp(cross(t - s, p[a] - s));**

**int d2 = dcmp(cross(t - s, p[b] - s));**

**if( d1 \* d2 != -1 )**

**return 0;**

**d1 = find(a, (b-a+n)%n, s, t, p, n);**

**d2 = find(b, (a-b+n)%n, s, t, p, n);**

**if( d1 > d2 ) swap(d1, d2);**

**lineCross(s, t, p[d1], p[d1+1], res1);**

**lineCross(s, t, p[d2], p[d2+1], res2);**

**res = cross(p[d2], res2) + cross(res2, res1) + cross(res1, p[d1]);**

**res += get(d1, d2);**

**return fabs(res);**

**}**

**//SGU 345**

**int main()**

**{**

**int n, m;**

**point s, t;**

**while( scanf( "%d", &n ) != EOF ) {**

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

**pnt[i].input();**

**n = andrew(n);**

**init(res, n);**

**double area = fabs(get(0, n)), tmp;**

**scanf("%d", &m);**

**while( m-- ) {**

**s.input(); t.input();**

**tmp = line\_cut\_con(s, t, res, n);**

**tmp = min(tmp, area - tmp);**

**printf("%.10lf\n", tmp \* 0.5);**

**}**

**}**

**return 0;**

**}**

**5.7 动态凸包**

**#define spit set<point>::iterator**

**//部分函数省略**

**struct point {**

**LL x, y;**

**double ang;**

**bool operator < (const point &b) const {**

**return ang < b.ang;**

**}**

**double angle(double X, double Y) {**

**return atan2(y - Y, x - X);**

**}**

**};**

**bool cmp( point a, point b ) {**

**return a.x < b.x || a.x == b.x && a.y < b.y;**

**}**

**set<point> st;**

**vector<point> vec;**

**LL area;**

**double X, Y;**

**void init(point a, point b, point c) { //abc不共线**

**st.clear();**

**X = (a.x + b.x + c.x + 0.0) / 3;**

**Y = (a.y + b.y + c.y + 0.0) / 3;**

**a.ang = a.angle(X, Y); st.insert(a);**

**b.ang = b.angle(X, Y); st.insert(b);**

**c.ang = c.angle(X, Y); st.insert(c);**

**area = cross(a, b) + cross(b, c) + cross(c, a);**

**if( area < 0 ) area = -area;**

**}**

**spit pre( spit it ) {**

**if( it == st.begin() ) it = st.end();**

**return --it;**

**}**

**spit nxt( spit it ) {**

**if( ++it == st.end() ) it = st.begin();**

**return it;**

**}**

**void update( point p ) {**

**p.ang = p.angle(X, Y);**

**spit it = pre(st.lower\_bound(p));**

**if( cross(\*nxt(it) - \*it, p - \*it) > 0 ) return ;**

**spit lft = it, rht = it;**

**while( cross(\*nxt(rht) - p, \*rht - p) >= 0 ) rht = nxt(rht);**

**while( cross(\*lft - p, \*pre(lft) - p) >= 0 ) lft = pre(lft);**

**it = lft;**

**while( it != rht ) area -= cross(\*it, \*(nxt(it))), it = nxt(it);**

**it = nxt(lft);**

**while( it != rht ) it = nxt(it), st.erase(pre(it));**

**area += cross(\*lft, p) + cross(p, \*rht);**

**st.insert(p);**

**}**

**//SGU 277**

**int main()**

**{**

**int n;**

**point a, b, c, p;**

**while( scanf( "%I64d%I64d", &a.x, &a.y ) == 2 ) {**

**area = 0; b.input(); c.input();**

**bool hav = false;**

**if( cross(b - a, c - a) != 0 ) init(a, b, c), hav = true;**

**else {**

**point t[3] = {a, b, c};**

**sort(t, t + 3, cmp);**

**a = t[0], b = t[2];**

**}**

**scanf( "%d", &n );**

**while( n-- ) {**

**p.input();**

**if( hav ) update(p);**

**else if( cross(b - a, p - a) == 0 ) {**

**point t[3] = {a, b, p};**

**sort(t, t + 3, cmp);**

**a = t[0], b = t[2];**

**}**

**else init(a, b, p), hav = true;**

**printf( "%I64d\n", area );**

**}**

**}**

**return 0;**

**}**

**5.8 任意多边形最大内切圆(点+线=3限制内切圆)**

**//省略部分函数**

**struct point {**

**double x, y;**

**point perp() {**

**return point(-y, x);**

**}**

**};**

**struct Line {**

**point s, t; bool seg;**

**Line(){}**

**Line(point s, point t, bool f = true):s(s), t(t), seg(f){}**

**bool intersectLine(const Line &L, point\* r = NULL) const {**

**point v1 = t - s, v2 = L.t - L.s;**

**point dS = L.s - s;**

**double D = v2.x \* v1.y - v1.x \* v2.y;**

**if (D == 0) return false;**

**double u1 = (dS.y \* v2.x - dS.x \* v2.y) / D;**

**double u2 = (dS.y \* v1.x - dS.x \* v1.y) / D;**

**if (r != NULL) \*r = s + v1 \* u1;**

**return ((!seg || (0 <= u1 && u1 <= 1))**

**&& (!L.seg || (0 <= u2 && u2 <= 1)));**

**}**

**};**

**double pointToLineDist(const point &p, const Line &L) {**

**point v = L.t - L.s;**

**double u = ((p.x - L.s.x) \* v.x + (p.y - L.s.y) \* v.y)**

**/ (v.x \* v.x + v.y \* v.y);**

**if (L.seg) u = max(min(u, 1.0), 0.0);**

**return (L.s + v \* u - p).len();**

**}**

**struct Quadr { // Ax^2 + By^2 + Cxy + Dx + Ey + F = 0**

**double A, B, C, D, E, F;**

**Quadr(){}**

**Quadr(double a, double b, double c, double d, double e, double f) {**

**A = a; B = b; C = c; D = d; E = e; F = f;**

**}**

**};**

**Line getBisector(const point &p1, const point &p2) {**

**point mid = (p1 + p2) / 2;**

**return Line(mid, mid + (p2 - p1).perp(), false);**

**}**

**Line getBisector(const Line &L1, const Line &L2) {**

**point v1 = L1.s - L1.t, v2 = L2.t - L2.s;**

**v1 = v1 / v1.len(); v2 = v2 / v2.len();**

**point v = (v1 + v2) / 2;**

**point p;**

**if (L1.intersectLine(L2, &p)) return Line(p, p + v, false);**

**else {**

**double u = ((L1.s.x - L2.s.x) \* v2.x + (L1.s.y - L2.s.y) \* v2.y)**

**/ (v2.x \* v2.x + v2.y \* v2.y);**

**p = L2.s + v2 \* u; v1 = (p - L1.s) / 2;**

**return Line(L1.s + v1, L1.t + v1);**

**}**

**}**

**Quadr getBisector(const point &p, const Line &L) {**

**point v = L.t - L.s; v = v / v.len(); v = v.perp();**

**double C = -v.x \* L.s.x - v.y \* L.s.y; // v.x \* x + v.y \* y + C = 0**

**return Quadr(1.0 - v.x \* v.x, 1.0 - v.y \* v.y, -2.0 \* v.x \* v.y, -2.0 \***

**(p.x + v.x \* C), -2.0 \* (p.y + v.y \* C), p.x \* p.x + p.y \* p.y - C \* C);**

**}**

**vector<point> intersect(const Line &L, const Quadr &Q) {**

**vector<point> V;**

**point v = L.t - L.s; v = v / v.len();**

**double A = Q.A \* v.x \* v.x + Q.B \* v.y \* v.y + Q.C \* v.x \* v.y;**

**double B = 2.0 \* (Q.A \* L.s.x \* v.x + Q.B \* L.s.y \* v.y)**

**+ Q.C \* (L.s.x \* v.y + L.s.y \* v.x) + Q.D \* v.x + Q.E \* v.y;**

**double C = Q.A \* L.s.x \* L.s.x + Q.B \* L.s.y \* L.s.y**

**+ Q.C \* L.s.x \* L.s.y + Q.D \* L.s.x + Q.E \* L.s.y + Q.F;**

**if (A == 0) {**

**if (B != 0.0) {**

**double u = -C/B;**

**V.push\_back(L.s + v \* u);**

**}**

**return V;**

**}**

**double D = B \* B - 4.0 \* A \* C;**

**if (D < 0.0) return V;**

**D = sqrt(D);**

**double u1 = (-B + D)/(2.0 \* A);**

**double u2 = (-B - D)/(2.0 \* A);**

**V.push\_back(L.s + v \* u1); V.push\_back(L.s + v \* u2);**

**return V;**

**}**

**int N;**

**point P[25];**

**double maxR;**

**double fitCircle(const point &p) {**

**if (!pointInPoly(p)) return 0.0;**

**double R = 1000000;**

**for (int i = 0; i < N; i++) {**

**int j = (i+1)%N;**

**R = min(R, pointToLineDist(p, Line(P[i], P[j])));**

**}**

**return R;**

**}**

**void check(const point &p1, const point &p2, const point &p3) {**

**point r;**

**if(getBisector(p1, p2).intersectLine(getBisector(p2, p3), &r))**

**maxR = max(maxR, fitCircle(r));**

**}**

**void check(const point &p1, const point &p2, const Line &L) {**

**vector<point> V = intersect(getBisector(p1, p2), getBisector(p1, L));**

**for(int i = 0; i < V.size(); i++) maxR = max(maxR, fitCircle(V[i]));**

**}**

**void check(const point &p, const Line &L1, const Line &L2) {**

**vector<point> V = intersect(getBisector(L1, L2), getBisector(p, L1));**

**for (int i = 0; i < V.size(); i++) maxR = max(maxR, fitCircle(V[i]));**

**}**

**void check(const Line &L1, const Line &L2, const Line &L3) {**

**point r;**

**if(getBisector(L1, L2).intersectLine(getBisector(L2, L3), &r))**

**maxR = max(maxR, fitCircle(r));**

**}**

**void solve() {**

**cin >> N;**

**for (int i = 0; i < N; i++) cin >> P[i].x >> P[i].y;**

**maxR = 0.0;**

**for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) {**

**if (i == j) continue;**

**for (int k = 0; k < N; k++) {**

**if (k == i || k == j) continue;**

**int i2 = (i+1)%N, j2 = (j+1)%N, k2 = (k+1)%N;**

**check(P[i], P[j], P[k]);**

**if (k2 != i && k2 != j)**

**check(P[i], P[j], Line(P[k], P[k2], 0));**

**if (k2 != i && j2 != i)**

**check(P[i], Line(P[j], P[j2], 0), Line(P[k], P[k2], 0));**

**check(Line(P[i], P[i2], 0), Line(P[j], P[j2], 0), Line(P[k], P[k2], 0));**

**}**

**}**

**printf( "%.2lf\n", maxR );**

**}**

**5.9多边形面积并**

**//输入点集为逆时针，输入后调用init()**

**struct polygon {**

**point p[500];**

**int sz;**

**void init() {**

**p[sz] = p[0];**

**}**

**}g[505];**

**pair<double, int> c[100000];**

**double segP( point a, point b, point c ) {**

**if( dcmp(b.x - c.x) )**

**return (a.x - b.x) / (c.x - b.x);**

**return (a.y - b.y) / (c.y - b.y);**

**}**

**double polyUnion( int n )**

**{**

**double sum = 0;**

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

**for( int ii = 0; ii < g[i].sz; ++ii ) {**

**int tot = 0;**

**c[tot++] = MP(0, 0);**

**c[tot++] = MP(1, 0);**

**for( int j = 0; j < n; ++j ) if( i != j )**

**for( int jj = 0; jj < g[j].sz; ++jj ) {**

**int d1 = dcmp(cross(g[i].p[ii+1] - g[i].p[ii],**

**g[j].p[jj] - g[i].p[ii]));**

**int d2 = dcmp(cross(g[i].p[ii+1] - g[i].p[ii],**

**g[j].p[jj+1] - g[i].p[ii]));**

**if( !d1 && !d2 ) {**

**point t1 = g[j].p[jj+1] - g[j].p[jj];**

**point t2 = g[i].p[ii+1] - g[i].p[ii];**

**if( dcmp( dot(t1, t2) ) > 0 && j < i ) {**

**c[tot++]=MP(segP(g[j].p[jj], g[i].p[ii], g[i].p[ii+1]), 1);**

**c[tot++]=MP(segP(g[j].p[jj+1],g[i].p[ii],g[i].p[ii+1]), -1);**

**}**

**}**

**else if( d1 >= 0 && d2 < 0 || d1 < 0 && d2 >= 0 ) {**

**double tc = cross(g[j].p[jj+1] - g[j].p[jj],**

**g[i].p[ii] - g[j].p[jj]);**

**double td = cross(g[j].p[jj+1] - g[j].p[jj],**

**g[i].p[ii+1] - g[j].p[jj]);**

**if( d2 < 0 )**

**c[tot++] = MP(tc / (tc - td), 1);**

**else c[tot++] = MP(tc / (tc - td), -1);**

**}**

**}**

**sort(c, c + tot);**

**double cur = min(max(c[0].first, 0.0), 1.0);**

**int sgn = c[0].second;**

**double s = 0;**

**for( int j = 1; j < tot; ++j ) {**

**double nxt = min(max(c[j].first, 0.0), 1.0);**

**if( !sgn ) s += nxt - cur;**

**sgn += c[j].second;**

**cur = nxt;**

**}**

**sum += cross(g[i].p[ii], g[i].p[ii+1]) \* s;**

**}**

**return sum / 2;**

**}**

**六：概率算法**

**6.1 最小球覆盖（模拟退火）**

**int cal( point t, int n ) {**

**int id = -1;**

**double r = 0;**

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

**double d = (p[i] - t).len();**

**if( d > r )**

**r = d, id = i;**

**}**

**return id;**

**}**

**double solve( int n ) {**

**double r = 0.0;**

**point t = point(0, 0, 0);**

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

**r = max(r, p[i].len());**

**double dlt = r;**

**while( dlt > eps ) {**

**int id = cal(t, n);**

**double d = (p[id] - t).len();**

**r = min(r, d);**

**t.x += (p[id].x - t.x) / d \* dlt;**

**t.y += (p[id].y - t.y) / d \* dlt;**

**t.z += (p[id].z - t.z) / d \* dlt;**

**dlt \*= 0.98;**

**}**

**return r;**

**}**

**6.2 平面费马点**

**double solve( int n ) {**

**point t = point(0, 0);**

**double r = 0;**

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

**r += p[i].len();**

**double dlt = 10000;**

**while( dlt > eps ) {**

**for( int i = 0; i < 30; ++i ) {**

**double ang = (rand() % 20000 \* pi) / 10000 - pi;**

**point k = point(t.x + dlt \* cos(ang), t.y + dlt \* sin(ang));**

**double rk = 0;**

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

**rk += (p[i] - k).len();**

**if( rk < r )**

**r = rk, t = k;**

**}**

**dlt \*= 0.98;**

**}**

**return r;**

**}**

**七：平面问题**

**7.1 半平面交**

**//直线用向量法p-v表示，半平面为直线左侧平面**

**struct line {**

**point p, v;**

**double ang;**

**line(){}**

**line( point p, point v ) :p(p), v(v) { ang = atan2(v.y, v.x); }**

**bool operator < ( const line &b ) const {**

**return ang < b.ang;**

**}**

**}L[mxn], q[mxn];**

**point p[mxn], poly[mxn];**

**point normal( point a ) {**

**double l = a.len();**

**return point( -a.y / l, a.x / l );**

**}**

**bool onleft( line l, point p ) {**

**return dcmp( cross( l.v, p - l.p ) ) > 0;**

**}**

**point lineinter( line a, line b ) {**

**point u = a.p - b.p;**

**double t = cross( b.v, u ) / cross( a.v, b.v );**

**return a.p + a.v \* t;**

**}**

**int halfplane( int n ) {**

**sort( L, L + n);**

**int head = 0, tail = 0;**

**q[tail] = L[0];**

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

**while( head < tail && !onleft( L[i], p[tail-1] ) ) --tail;**

**while( head < tail && !onleft( L[i], p[head] ) ) ++head;**

**q[++tail] = L[i];**

**if( dcmp( cross( q[tail].v, q[tail-1].v ) ) == 0 ) {**

**--tail;**

**if( onleft( q[tail], L[i].p ) ) q[tail] = L[i];**

**}**

**if( head < tail )**

**p[tail-1] = lineinter( q[tail-1], q[tail] );**

**}**

**while( head < tail && !onleft( q[head], p[tail-1] ) ) --tail;**

**if( tail - head <= 1 ) return 0;**

**p[tail] = lineinter( q[tail], q[head] );**

**int m = 0;**

**for( int i = head; i <= tail; ++i ) poly[m++] = p[i];**

**return m;**

**}**

**7.2 PSLG平面直线图**

**#define Polygon vector<point>**

**//省略部分函数**

**struct point {**

**double x, y;**

**bool operator < (const point &b) const {**

**return dcmp(x - b.x) < 0 || dcmp(x - b.x) == 0 && dcmp(y - b.y) < 0;**

**}**

**bool operator == (const point &b) const {**

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

**}**

**};**

**point LineCross(point &P, point &Pv, point &Q, point &Qw) {**

**point u = P - Q;**

**point v = Pv - P, w = Qw - Q;**

**double t = cross(w, u) / cross(v, w);**

**return P + v \* t;**

**}**

**bool SegInter(point &a1, point &a2, point &b1, point &b2) {**

**double c1 = dcmp(cross(a2 - a1,b1 - a1));**

**double c2 = dcmp(cross(a2 - a1,b2 - a1));**

**double c3 = dcmp(cross(b2 - b1,a1 - b1));**

**double c4 = dcmp(cross(b2 - b1,a2 - b1));**

**return c1 \* c2 < 0 && c3 \* c4 < 0;**

**}**

**bool ponseg(point p, point a, point b) {**

**return dcmp(cross(a - p, b - p)) == 0**

**&& dcmp(dot(a - p, b - p)) < 0;**

**}**

**double PolygonArea(Polygon poly) {**

**double area = 0;**

**int n = poly.size();**

**for( int i = 1; i < n - 1; ++i )**

**area += cross(poly[i] - poly[0], poly[(i+1)%n] - poly[0]);**

**return area / 2;**

**}**

**struct Edge {**

**int from, to;**

**double ang;**

**Edge(){}**

**Edge(int f, int t, double a):from(f), to(t), ang(a){}**

**};**

**const int mxn = 10000 + 10; // 最大边数**

**struct PSLG {**

**int n, m, face\_cnt;**

**double x[mxn], y[mxn];**

**vector<Edge> edges;**

**vector<int> G[mxn];**

**int vis[mxn\*2]; // 每条边是否已经访问过**

**int left[mxn\*2]; // 左面的编号**

**int prev[mxn\*2];**

**// prev 相同起点的上一条边（即顺时针旋转碰到的下一条边）d 编号**

**vector<Polygon> faces;**

**double area[mxn]; // 每个polygon的面积**

**void init(int n) {**

**this->n = n;**

**for( int i = 0; i < n; ++i ) G[i].clear();**

**edges.clear();**

**faces.clear();**

**}**

**// 有向线段from->to的极角**

**double getAngle(int from, int to) {**

**return atan2(y[to] - y[from], x[to] - x[from]);**

**}**

**void AddEdge(int from, int to) {**

**edges.push\_back(Edge(from, to, getAngle(from, to)));**

**edges.push\_back(Edge(to, from, getAngle(to, from)));**

**m = edges.size();**

**G[from].push\_back(m - 2);**

**G[to].push\_back(m - 1);**

**}**

**// 找出faces并计算面积**

**void Build() {**

**for( int u = 0; u < n; ++u ) {**

**// 给从u出发的各条边按极角排序**

**int d = G[u].size();**

**for( int i = 0; i < d; ++i )**

**for( int j = i+1; j < d; ++j ) //假设从每点出发的线段不多**

**if(edges[G[u][i]].ang > edges[G[u][j]].ang)**

**swap(G[u][i], G[u][j]);**

**//必要时把edges拿出去，写索引sort**

**for(int i = 0; i < d; i++)**

**prev[G[u][(i+1)%d]] = G[u][i];**

**}**

**memset(vis, 0, sizeof(vis));**

**face\_cnt = 0;**

**for( int u = 0; u < n; ++u )**

**for( int i = 0; i < G[u].size(); ++i ) {**

**int e = G[u][i];**

**if( !vis[e] ) { // 逆时针找圈**

**face\_cnt++;**

**Polygon poly;**

**for(;;) {**

**vis[e] = 1; left[e] = face\_cnt;**

**int from = edges[e].from;**

**poly.push\_back(point(x[from], y[from]));**

**e = prev[e^1];**

**if(e == G[u][i]) break;**

**//assert(vis[e] == 0);**

**}**

**faces.push\_back(poly);**

**}**

**}**

**for(int i = 0; i < faces.size(); i++)**

**area[i] = PolygonArea(faces[i]);**

**}**

**};**

**PSLG g;**

**const int mxp = 100 + 5;**

**int n, c;**

**point P[mxp];**

**point V[mxp\*(mxp-1)/2+mxp];**

**// 在V数组里找到点p**

**int ID(point p) {**

**return lower\_bound(V, V + c, p) - V;**

**}**

**// 假定poly没有相邻点重合，只需删除三点共线**

**Polygon simplify(const Polygon &poly) {**

**Polygon ans;**

**int n = poly.size();**

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

**point a = poly[i];**

**point b = poly[(i+1)%n];**

**point c = poly[(i+2)%n];**

**if(dcmp(cross(a-b, c-b)) != 0)**

**ans.push\_back(b);**

**}**

**return ans;**

**}**

**void build\_graph() {**

**c = n;**

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

**V[i] = P[i];**

**vector<double> dist[mxp]; // dist[i][j]是第i条线段上的第j个点离起点（P[i]）的距离**

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

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

**if(SegInter(P[i], P[(i+1)%n], P[j], P[(j+1)%n])) {**

**point p = LineCross(P[i], P[(i+1)%n], P[j], P[(j+1)%n]);**

**V[c++] = p;**

**dist[i].push\_back((p - P[i]).len());**

**dist[j].push\_back((p - P[j]).len());**

**}**

**sort(V, V + c);**

**c = unique(V, V + c) - V;**

**g.init(c); // c是平面图的点数**

**for( int i = 0; i < c; ++i ) {**

**g.x[i] = V[i].x;**

**g.y[i] = V[i].y;**

**}**

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

**point v = P[(i+1)%n] - P[i];**

**double len = v.len();**

**dist[i].push\_back(0);**

**dist[i].push\_back(len);**

**sort(dist[i].begin(), dist[i].end());**

**int sz = dist[i].size();**

**for( int j = 1; j < sz; ++j ) {**

**point a = P[i] + v \* (dist[i][j-1] / len);**

**point b = P[i] + v \* (dist[i][j] / len);**

**if(a == b) continue;**

**g.AddEdge(ID(a), ID(b));**

**}**

**}**

**g.Build();**

**Polygon poly;**

**for( int i = 0; i < g.faces.size(); ++i ) if(g.area[i] < 0) {**

**// 对于连通图，惟一一个面积小于零的面是无限面**

**poly = g.faces[i];**

**reverse(poly.begin(), poly.end());**

**// 对于内部区域来说，无限面多边形的各个顶点是顺时针的**

**poly = simplify(poly); // 无限面多边形上可能会有相邻共线点**

**break;**

**}**

**int m = poly.size();**

**printf("%d\n", m);**

**// 挑选坐标最小的点作为输出的起点**

**int start = 0;**

**for( int i = 0; i < m; ++i )**

**if(poly[i] < poly[start])**

**start = i;**

**for( int i = start; i < m; ++i )**

**printf("%.4lf %.4lf\n", poly[i].x, poly[i].y);**

**for( int i = 0; i < start; ++i )**

**printf("%.4lf %.4lf\n", poly[i].x, poly[i].y);**

**}**

**//LA3218 自交多边形找不自交边界**

**int main()**

**{**

**while(scanf("%d", &n) == 1 && n) {**

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

**int x, y;**

**scanf("%d%d", &x, &y);**

**P[i] = point(x, y);**

**}**

**build\_graph();**

**}**

**return 0;**

**}**

**八：三维几何**

**8.1 三维叉乘**

**point cross( point a, point b ) {**

**point res;**

**res.x = a.y \* b.z - b.y \* a.z;**

**res.y = a.z \* b.x - b.z \* a.x;**

**res.z = a.x \* b.y - b.x \* a.y;**

**return res;**

**}**

**8.2 三维旋转矩阵**

**//向量p绕向量v逆时针旋转af弧度，多重多种旋转可以用矩阵快速幂加速**

**point rot( point p, point v, double af ) {**

**af = af \* pi / 180;**

**double c = cos( af ), s = sin( af );**

**double l = v.len();**

**double x = v.x / l, y = v.y / l, z = v.z / l;**

**double a[3][3] = {**

**{ x \* x + ( 1 - x \* x ) \* c, x \* y \* ( 1 - c ) - z \* s, x \* z \* ( 1 - c ) + y \* s },**

**{ y \* x \* ( 1 - c ) + z \* s, y \* y + ( 1 - y \* y ) \* c, y \* z \* ( 1 - c ) - x \* s },**

**{ z \* x \* ( 1 - c ) - y \* s, z \* y \* ( 1 - c ) + x \* s, z \* z + ( 1 - z \* z ) \* c }**

**};**

**point res = point (**

**p.x \* a[0][0] + p.y \* a[0][1] + p.z \* a[0][2],**

**p.x \* a[1][0] + p.y \* a[1][1] + p.z \* a[1][2],**

**p.x \* a[2][0] + p.y \* a[2][1] + p.z \* a[2][2]**

**);**

**return res;**

**}**

**8.3 三维旋转模型**

**1.将三维凸包一个面贴在地上**

**plane pl = vp[i]; //凸包的一个面**

**point z = point( 0, 0, 1 ); //z轴**

**point x = cross( pl.f, z ); //旋转轴**

**double af = angle( z, pl.f ); //旋转角度**

**把每个点都rot(p[], x, af)**

**2.多重旋转矩阵加速**

**旋转方式**

**translate tx ty tz**

**Everything in (x, y, z) must be moved to ( x+tx, y+ty, z+tz)**

**scale a b c**

**Everything in (x, y, z) will be moved to (ax, by, cz)**

**rotate a b c d**

**Everything 绕向量v(a, b, c) 逆时针旋转d弧度**

**定义一系列上述旋转序列，循环k次，求点p旋转后的位置**

**每种操作可以得到转化为一个矩阵，把旋转序列的矩阵都乘起来，再求k次幂得到矩阵tmp**

**设点p最终的位置是点o**

**则有[p.x, p.y, p.z, 1] \* tmp[4][4] = [o.x, o.y, o.z, not\_use]**

**下述矩阵未赋值的位置均为0**

**void trans( point p ) { // p = point(tx, ty, tz)**

**m[0][0] = m[1][1] = m[2][2] = m[3][3] = 1;**

**m[3][0] = p.x, m[3][1] = p.y, m[3][2] = p.z;**

**}**

**void scal( point p ) { // p = point(a, b, c)**

**m[0][0] = p.x, m[1][1] = p.y, m[2][2] = p.z, m[3][3] = 1;**

**}**

**void rot( point v, double af ) {**

**m[][]左上角3\*3赋值为8.2中rot(v, af)的矩阵a**

**m[3][3] = 1;**

**}**

**8.4 三维凸包相关**

**const int mxn = 550;**

**const double eps = 1e-8;**

**int n;**

**struct face {**

**int a, b, c;**

**point v;**

**//表示该面是否属于最终凸包上的面**

**bool ok;**

**void init() {**

**v = cross( p[b] - p[a], p[c] - p[a] );**

**}**

**};**

**struct CH3D {**

**int num; //凸包表面的三角形数**

**face F[8\*mxn]; //凸包表面的三角形**

**int g[mxn][mxn]; //凸包表面的边**

**double area( point a, point b, point c ) {**

**return cross( b - a, c - a ).len() / 2;**

**}**

**//四面体有向体积**

**double volume( point a, point b, point c, point d ) {**

**return dot( cross( b - a, c - a ), d - a ) / 6;**

**}**

**//点与面法向量方向关系 1同向 0点在面上 -1反向**

**int pside( point pt, face f ) {**

**f.init();**

**return dcmp( dot( f.v, pt - p[f.a] ) );**

**}**

**void deal( int pt, int a, int b ) {**

**int f = g[a][b]; //搜索与该边相邻的另一个平面**

**face add;**

**if( F[f].ok ) {**

**if( pside( p[pt], F[f] ) == 1 )**

**dfs( pt, f );**

**else {**

**add.a = b;**

**add.b = a;**

**add.c = pt; //顺序要成右手系**

**add.ok = true;**

**g[pt][b] = g[a][pt] = g[b][a] =num;**

**F[num++] = add;**

**}**

**}**

**}**

**void dfs( int pt, int now ) { //递归搜索所有应该从凸包内删除的面**

**F[now].ok = 0;**

**deal( pt, F[now].b, F[now].a );**

**deal( pt, F[now].c, F[now].b );**

**deal( pt, F[now].a, F[now].c );**

**}**

**bool same( int s, int t ) {**

**point &a = p[F[s].a];**

**point &b = p[F[s].b];**

**point &c = p[F[s].c];**

**return dcmp( volume( a, b, c, p[F[t].a] ) ) == 0 &&**

**dcmp( volume( a, b, c, p[F[t].b] ) ) == 0 &&**

**dcmp( volume( a, b, c, p[F[t].c] ) ) == 0;**

**}**

**void create() { //构建三维凸包**

**face add;**

**num = 0;**

**if( n < 4 ) return; //hehe**

**bool flag = true;**

**for( int i = 1; i < n; ++i ) { //前两点不共点**

**if( dcmp( (p[0] - p[i]).len() ) > 0 ) {**

**swap( p[1], p[i] );**

**flag = false;**

**break;**

**}**

**}**

**if( flag ) return;**

**flag = true;**

**for( int i = 2; i < n; ++i ) { //前三点不共线**

**if( dcmp( cross( p[0] - p[1], p[i] - p[1] ).len() ) != 0 ) {**

**swap( p[2] ,p[i] );**

**flag =false;**

**break;**

**}**

**}**

**if( flag )return;**

**flag = true;**

**for( int i = 3; i < n; ++i ) { //前四点不共面**

**if( dcmp( volume( p[0], p[1], p[2], p[i] ) ) != 0 ) {**

**swap( p[3], p[i] );**

**flag = false;**

**break;**

**}**

**}**

**if( flag ) return;**

**for( int i = 0; i < 4; ++i ) {**

**add.a = (i + 1) % 4;**

**add.b = (i + 2) % 4;**

**add.c = (i + 3) % 4;**

**add.ok = true;**

**if( pside( p[i], add ) == 1 ) swap( add.b, add.c );**

**g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;**

**F[num++] = add;**

**}**

**for( int i = 4; i < n; ++i ) {**

**for( int j = 0; j < num; ++j) {**

**if( F[j].ok && pside( p[i], F[j] ) == 1 ) {**

**dfs( i, j );**

**break;**

**}**

**}**

**}**

**int tmp = num; num = 0;**

**for( int i = 0; i < tmp; ++i )**

**if( F[i].ok )**

**F[num++] = F[i];**

**}**

**double calarea() { //表面积**

**double res=0;**

**if( n == 3 )**

**return area( p[0], p[1], p[2] );**

**for( int i = 0; i < num; ++i )**

**res += area( p[F[i].a], p[F[i].b], p[F[i].c] );**

**return res;**

**}**

**double calvol() { //体积**

**double res = 0;**

**point o( 0, 0, 0 );**

**for( int i = 0; i < num; ++i )**

**res += volume( p[F[i].a], p[F[i].b], p[F[i].c], o );**

**return fabs( res );**

**}**

**//表面多边形个数**

**int polygon() {**

**int res = 0;**

**for( int i = 0; i < num; ++i ) {**

**int flag = 1;**

**for( int j = 0; j < i; ++j ) {**

**if( same( i, j ) ) {**

**flag = 0;**

**break;**

**}**

**}**

**res += flag;**

**}**

**return res;**

**}**

**//三维凸包重心**

**point barycenter()**

**{**

**point ans( 0, 0, 0 ), o( 0, 0, 0 );**

**double all = 0;**

**for( int i = 0; i < num; ++i ) {**

**double vol = volume( p[F[i].a], p[F[i].b], p[F[i].c], o) \* 6;**

**ans = ans + ( o + p[F[i].a] + p[F[i].b] + p[F[i].c]) \* vol / 4;**

**all += vol;**

**}**

**ans = ans / all;**

**return ans;**

**}**

**double ptoface( point pt, int i ) {**

**face tmp;**

**tmp.a = F[i].a; tmp.b = F[i].b; tmp.c = F[i].c;**

**tmp.init();**

**return fabs( dot( pt - p[tmp.a], tmp.v ) ) / tmp.v.len();**

**}**

**};**

**CH3D hull; //内有大数组，不宜定义在函数内**

**int main()**

**{**

**while( scanf( "%d", &n ) == 1 ) {**

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

**p[i].input();**

**hull.create();**

**point pt = hull.barycenter();**

**double opt = 1e20;**

**for( int i = 0; i < hull.num; ++i )**

**opt = min( opt, hull.ptoface( pt, i ) );**

**printf( "%.3lf\n", opt );**

**}**

**return 0;**

**}**

**8.5 三维光线反射**

**1.平面反射**

**射线起点s，方向v，平面p0-n**

**void reflect(point s, point v, point p0, point n, point &rs, point &rv) {**

**rs = LinePlaneInter(s, s + v, p0, n);**

**point tmp = p\_plane\_q(s, p0, n);**

**rv = rs - tmp;  
}**

**2.球面反射**

**射线起点s，方向v，球心p，半径r**

**bool reflect(point s, point v, point p, double r, point &rs, point &rv) {**

**double a = dot(v, v);**

**double b = dot(s - p, v) \* 2;**

**double c = dot(s - p, s - p) - r \* r;**

**double dlt = b \* b - 4 \* a \* c;**

**if( dlt < 0 ) return false;**

**double t = (-b - sqrt(dlt)) / a / 2;**

**rs = s + v \* t;**

**point tn = p - rs;**

**rv = v - tn \* (dot(v, tn) \* 2 / dot(tn, tn));**

**return true;  
}**

**8.6 点到直线距离**

**double ptoline( point p, point a, point b ) {**

**return (cross( p - a, b - a )).len() / dis( a, b );**

**}**

**8.7 点到线段距离**

**double ptoseg( point p, point a, point b ) {**

**if( dcmp(dot( p - a, b - a )) < 0 ) return dis( p, a );**

**if( dcmp(dot( p - b, a - b )) < 0 ) return dis( p, b );**

**return (cross( p - a, b - a )).len() / dis( a, b );**

**}**

**8.8 两直线距离**

**//n.len()为0说明直线平行**

**double LineDis( point a, point b, point c, point d ) {**

**point n = cross(a - b, c - d);**

**if( dcmp(n.len()) == 0 ) return ptoline(a, c, d);**

**return fabs(dot(a - c, n)) / n.len();**

**}**

**8.9 两线段距离**

**double SegDis( point a, point b, point c, point d ) {**

**point n = cross(a - b, c - d);**

**if( dcmp(n.len()) != 0 ) {**

**point cc = ptoplane(c, a, n);**

**point dd = ptoplane(d, a, n);**

**point res;**

**if( SegCross(a, b, cc, dd, res) == 1 )**

**return LineDis(a, b, c, d);  
 }**

**double ret = ptoseg(a, c, d);**

**ret = min(ret, ptoseg(b, c, d));**

**ret = min(ret, ptoseg(c, a, b));**

**ret = min(ret, ptoseg(d, a, b));**

**return ret;  
}**

**8.10 直线相交判定**

**类型 返回 res**

**--------------------------------------**

**1. 不相交（平行） 0 不变**

**2. 规范相交 1 交点**

**3. 非规范相交（重合） 2 不变**

**4. 异面不相交 3 不变**

**int LineCross( point a, point b, point c, point d, point &res ) {**

**point n = cross(a - b, c - d);**

**if( dcmp(n.len()) == 0 ) {**

**if( dcmp(cross(a - b, c - b).len()) == 0 ) return 2;**

**return 0;  
 }**

**if( dcmp(ptoline(a, c, d)) == 0 ) {res = a; return 1;}**

**if( dcmp(ptoline(b, c, d)) == 0 ) {res = b; return 1;}**

**if( dcmp(ptoline(c, a, b)) == 0 ) {res = c; return 1;}**

**if( dcmp(ptoline(d, a, b)) == 0 ) {res = d; return 1;}**

**if( dcmp(dot( cross( b - a, c - a ), d - a )) != 0 ) return 3;**

**n = d + n;**

**point f = cross(d - c, n - c);**

**double t = dot(f, c - a) / dot(f, b - a);**

**res = a + (b - a) \* t;**

**return 1;**

**}**

**8.11 线段相交判定**

**类型 返回 res**

**--------------------------------------**

**1. 不相交 0 不变**

**2. 规范相交 1 交点**

**3. 非规范相交 2 不变**

**int SegCross(point a, point b, point c, point d, point &res) {**

**int k = LineCross(a, b, c, d, res);**

**if( k == 0 || k == 3 ) return 0;**

**if( k == 1 ) {**

**double d1 = dot(a - res, b - res);**

**double d2 = dot(c - res, d - res);**

**if( d1 < 0 && d2 < 0 ) return 1;**

**if( d1 == 0 && d2 <= 0 || d2 == 0 && d1 <= 0 ) return 2;**

**return 0;  
 }**

**if( dot(a - c, b - c) <= 0 || dot(a - d, b - d) <= 0**

**|| dot(c - a, d - a) <= 0 || dot(c - b, d - b) <= 0)**

**return 2;**

**return 0;  
}**

**8.12 点关于直线的对称点**

**point p\_line\_q(point p, point a, point b) {**

**point k = cross(b - a, p - a);**

**if( dcmp(k.len()) == 0 ) return p;**

**k = cross(k, b - a);**

**return p\_plane\_q(p, a, k);  
}**

**8.13 点到平面距离**

**//点p到平面p0-n的距离，不加fabs是有向距离**

**double distoplane(point p, point p0, point n) {**

**return fabs(dot(p - p0, n)) / n.len();  
}**

**8.14 点在平面投影**

**//点p在平面p0-n上的投影**

**point ptoplane(point p, point p0, point n) {**

**double d = dot(p - p0, n) / n.len();**

**return p - n \* d;  
}**

**8.15 点关于平面的对称点**

**//点p关于平面p0-n的对称点**

**point p\_plane\_q(point p, point p0, point n) {**

**double d = 2 \* dot(p - p0, n) / n.len();**

**return p - n \* d;**

**}**

**8.16 直线与平面交点**

**//直线p1-p2到平面p0-n的交点**

**//分母(dot(n, p2 - p1))为0说明直线与平面平行或直线在平面上**

**point LinePlaneInter(point p1, point p2, point p0, point n) {**

**point v = p2 - p1;**

**double t = dot(n, p0 - p1) / dot(n, p2 - p1);**

**return p1 + v \* t;  
}**

**8.17 线段与平面交点**

**//线段p1-p2到平面p0-n的交点，返回0说明无交点**

**//分母(dot(n, p2 - p1))为0说明线段与平面平行或直线在平面上**

**int SegPlaneInter(point p1, point p2, point p0, point n, point &res) {**

**point v = p2 - p1;**

**double t = dot(n, p0 - p1) / dot(n, p2 - p1);**

**if( dcmp(t) < 0 || dcmp(t - 1) > 0 ) return 0;**

**res = p1 + v \* t;**

**return 1;**

**}**

**8.18 直线与平面位置关系判定**

**//直线p1-p2与平面p0-n的位置关系**

**//0:相交 1:平行 2:垂直**

**int LineAndPlane(point p1, point p2, point p0, point n) {**

**point v = p2 - p1;**

**if( dcmp(dot(v, n)) == 0 ) return 1;**

**if( dcmp(cross(v, n).len()) == 0 ) return 2;**

**return 0;  
}**

**8.19 两平面位置关系判定**

**//平面p0-n0和p1-n1的位置关系**

**//0:有唯一交线 1:两平面垂直 2:两平面重合 3:两平面平行不重合**

**int PlaneAndPlane(point p0, point n0, point p1, point n1) {**

**if( dcmp(dot(n0, n1)) == 0 ) return 1;**

**if( dcmp(cross(n0, n1).len()) == 0 ) {**

**if( dcmp(dot(n0, p1 - p0)) == 0 ) return 2;**

**return 3;  
 }**

**return 0;  
}**

**8.20 平面交线**

**//平面p0-n0和p1-n1的交线，仅知道这4个量的时候，返回直线是向量式**

**bool PlaneCross(point p0, point n0, point p1, point n1, point &s, point &v) {**

**v = cross(n0, n1);**

**if( dcmp(v.len()) == 0 ) return false;**

**point tmp = p0 + rot(n0, v, 90);**

**s = LinePlaneInter(p0, tmp, p1, n1);**

**return true;  
}**

**8.21 平面距离**

**//平面p0-n0和p1-n1的距离**

**double PlaneDis(point p0, point n0, point p1, point n1) {**

**if( PlaneAndPlane(p0, n0, p1, n1) != 3 ) return 0;**

**return fabs(dot(p1 - p0, n0)) / n0.len();  
}**

**8.22 点在空间三角形内判定**

**//判断点p是否在△abc中，包括边界**

**bool PointInTri(point p, point a, point b, point c) {**

**double area0 = cross(b - a, c - a).len();**

**double area1 = cross(a - p, b - p).len();**

**double area2 = cross(b - p, c - p).len();**

**double area3 = cross(c - p, a - p).len();**

**return dcmp(area1 + area2 + area3 - area0) == 0;  
}**

**8.23 线段和空间三角形的位置关系**

**//线段p1-p2是否与三角形abc相交**

**bool SegTriInter(point p1, point p2, point a, point b, point c, point &res) {**

**point n = cross(b - a, c - a);**

**if( dcmp(dot(n, p2 - p1)) == 0 ) return false;**

**//线段与三角形所在平面平行或重合，如果这种情况也算相交再求线段交点即可**

**double t = dot(n, a - p1) / dot(n, p2 - p1);**

**if( dcmp(t) < 0 || dcmp(t - 1) > 0 ) return false;**

**res = p1 + (p2 - p1) \* t;**

**return PointInTri(res, a, b, c);  
}**

**8.24 经纬度坐标转笛卡尔坐标**

**//lat 纬度 -90 ~ 90 lng 经度 -180 ~ 180 R 球体半径**

**void get(double lat, double lng, double &x, double &y, double &z) {**

**lat = lat \* pi / 180;**

**lng = lng \* pi / 180;**

**x = R \* cos(lat) \* cos(lng);**

**y = R \* cos(lat) \* sin(lng);**

**z = R \* sin(lat);  
}**

**8.25 球面距离**

**//ab是笛卡尔坐标**

**double cal(point a, point b, double R) {**

**double d = (a - b).len();**

**return 2 \* R \* asin(d/(2\*R));  
}**

**九：数据结构优化算法**

**9.1 K-D树**

**int K = 2; //维数**

**struct kdNode {**

**LL x[5];**

**int div, id;**

**};//优先队列里保存的pair带有点id，有了id干什么都随便了**

**int cmpNo;**

**int cmp( kdNode a, kdNode b ) {**

**return a.x[cmpNo] < b.x[cmpNo];**

**}**

**LL dis2( kdNode& a, kdNode& b ) {**

**LL res = 0;**

**for( int i = 0; i < K; ++i )**

**res += (a.x[i] - b.x[i]) \* (a.x[i] - b.x[i]);**

**return res;**

**}**

**void buildKD( int l, int r, kdNode\* p, int d ) {**

**if( l > r ) return;**

**int m = (l + r) >> 1;**

**cmpNo = d;**

**nth\_element( p + l, p + m, p + r + 1, cmp);**

**p[m].div = d;**

**buildKD( l, m - 1, p, (d + 1) % K );**

**buildKD( m + 1, r, p, (d + 1) % K );**

**}**

**//n个点 编号0 ~ n-1，建树调用buildKD(0,n-1,kp,0); kp是kdNode点集**

**priority\_queue<pair<LL,int> > Q; //(距离平方，点的id)**

**void KNN( int l, int r, kdNode tar, kdNode\* p, int k ) {**

**if( l > r ) return;**

**int m = (l + r) >> 1;**

**pair<LL,int> v = MP(dis2(p[m], tar), p[m].id);**

**if( Q.size() == k && v < Q.top() ) Q.pop();**

**if( Q.size() < k ) Q.push(v);**

**LL t = tar.x[ p[m].div ] - p[m].x[ p[m].div ];**

**if( t <= 0 ) {**

**KNN(l, m - 1, tar, p, k);**

**if( Q.top().first > t \* t )**

**KNN( m + 1, r, tar, p, k);**

**}**

**else if( t > 0 ) {**

**KNN( m + 1, r, tar, p, k);**

**if( Q.top().first > t \* t )**

**KNN( l, m - 1, tar, p, k);**

**}**

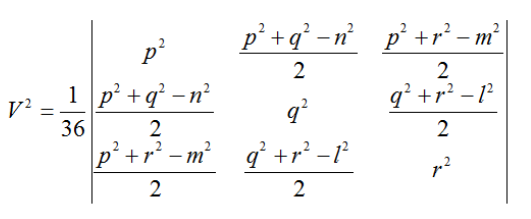
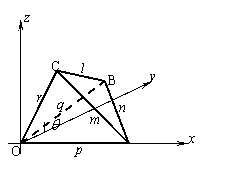
**}**

**//调用KNN(0,n-1,tar,kp,k) 查找tar点的k个临近点，存在Q中**

**十：其他**

**10.1 欧拉四面体公式**

**建立x，y，z直角坐标系。设A、B、C坐标分别为(a1,b,1,c1),(a2,b2,c2),(a3,b3,c3),四面体O-ABC的六条棱长分别为l,m,n,p,q,r (有向体积，注意取fabs)**



**10.2 simpson数值积分**

**double f(double x) {**

**return y = //函数值**

**}**

**double simpson(double x, double y) {**

**return (f(x) + f(y) + f((x+y) / 2) \* 4.0) / 6.0 \* (y - x);**

**}**

**double rsimpson(double x, double y) { //积分区域是[x,y]**

**double m = (x + y) / 2;**

**double res = simpson(x, y), l = simpson(x, m), r = simpson(m, y);**

**if( fabs(res-l-r) < eps\*15 )**

**return res;**

**return rsimpson(x, m) + rsimpson(m, y);**

**}**

**10.3 常用积分公式**

**1.弧长公式**

**有弧y = f(x)，则弧长L = ∫sqrt(1 + y’^2)**

**2.第一类曲线积分**

**设函数f(x,y)在曲线L上有定义，L的参数方程为**

**x = x(t), y = y(t)**

**则∫f(x,y)ds = ∫f[x(t),y(t)] \* sqrt(x’(t)^2 + y’(t)^2) dt**

**3.不定积分**

**1）∫kdx = kx+c**

**2）∫x^udx = (x^(u+1))/(u+1)+c**

**3）∫1/xdx = ln|x|+c**

**4) ∫a^xdx = (a^x)/lna+c**

**5）∫e^xdx = e^x+c**

**6）∫sinxdx = -cosx+c**

**7）∫cosxdx = sinx+c**

**8）∫1/(cosx)^2dx = tanx+c**

**9）∫1/(sinx)^2dx = -cotx+c**

**10）∫1/√(a^2-x^2)dx = arcsin(x/a)+c**

**11）∫1/(a^2+x^2)dx = 1/a\*arctan(x/a)+c**

**12）∫1/(a^2-x^2)dx = (1/(2a))ln|(a+x)/(a-x)|+c**

**13）∫secxdx = ln|secx+tanx|+c**

**14) ∫sec^2 x dx = tanx+c;**

**15) ∫shx dx = chx+c;**

**16) ∫chx dx = shx+c;**

**17) ∫thx dx = ln(chx)+c;**

**18) ∫1/(1+x^2) dx = arctanx+c**

**19) ∫1/√(1-x^2) dx = arcsinx+c**

**20) ∫tanx dx = -In|cosx|+c**

**21) ∫cotx dx = In|sinx|+c**

**22) ∫secx dx = In|secx+tanx|+c**

**23) ∫cscx dx = In|cscx-cotx|+c**

**24) ∫1/√(x^2+a^2) dx = In(x+√(x^2+a^2))+c**

**25) ∫1/√(x^2-a^2) dx = |In(x+√(x^2-a^2))|+c**

**10.4 三角函数**

**稀有函数**

**正割(sec)等于斜边比邻边；secA = c/b = 1 / cosA**

**余割(csc)等于斜边比对边。cscA = c/a = 1 / sinA**

**sinh / 双曲正弦：sh(x) = [e^x - e^(-x)] / 2**

**cosh / 双曲余弦：ch(x) = [e^x + e^(-x)] / 2**

**tanh / 双曲正切：th(x) = sh(x) / ch(x)=[e^x - e^(-x)] / [e^x + e^(-x)]**

**coth / 双曲余切：coth(x) = ch(x) / sh(x) = [e^x + e^(-x)] / [e^(x) - e^(-x)]**

**sech / 双曲正割：sech(x) = 1 / ch(x) = 2 / [e^x + e^(-x)]**

**csch / 双曲余割：csch(x) = 1 / sh(x) = 2 / [e^x - e^(-x)]**

**两角和与差的三角函数：**

**cos(α + β) = cosα·cosβ - sinα·sinβ**

**cos(α - β) = cosα·cosβ + sinα·sinβ**

**sin(α + β) = sinα·cosβ + cosα·sinβ**

**sin(α - β) = sinα·cosβ - cosα·sinβ**

**tan(α + β) = (tanα + tanβ) / (1 - tanα·tanβ)**

**tan(α - β) = (tanα - tanβ) / (1 + tanα·tanβ)**

**二倍角公式：**

**sin(2α) = 2sinα·cosα**

**cos(2α) = cos^2(α)-sin^2(α) = 2cos^2(α)-1 = 1-2sin^2(α)**

**tan(2α) = 2tanα/[1-tan^2(α)]**

**倍角公式：**

**sin3α = 3sinα - 4sin^3(α)**

**cos3α = 4cos^3(α) - 3cosα**

**半角公式：**

**sin^2(α/2) = (1-cosα)/2**

**cos^2(α/2) = (1+cosα)/2**

**tan^2(α/2) = (1-cosα)/(1+cosα)**

**tan(α/2) = sinα/(1+cosα) = (1-cosα)/sinα**

**万能公式：**

**半角的正弦、余弦和正切公式（降幂扩角公式）**

**sinα = 2tan(α/2) / [1+tan^2(α/2)]**

**cosα = [1-tan^2(α/2)] / [1+tan^2(α/2)]**

**tanα = 2tan(α/2) / [1-tan^2(α/2)]**

**积化和差公式：**

**sinα·cosβ = (1/2)[sin(α+β) + sin(α-β)]**

**cosα·sinβ = (1/2)[sin(α+β) - sin(α-β)]**

**cosα·cosβ = (1/2)[cos(α+β) + cos(α-β)]**

**sinα·sinβ = -(1/2)[cos(α+β) - cos(α-β)]**

**和差化积公式：**

**sinα + sinβ = 2sin[(α+β)/2]cos[(α-β)/2]**

**sinα – sinβ = 2cos[(α+β)/2]sin[(α-β)/2]**

**cosα + cosβ = 2cos[(α+β)/2]cos[(α-β)/2]**

**cosα – cosβ = -2sin[(α+β)/2]sin[(α-β)/2]**