# ACM/ICPC 比赛资料

# 计算几何

## 目录

缺：线段相交 ..... 5

几何公式 ..... 5

三角公式 Trigonometric Function Formula ..... 5

Product To Sum ..... 5

Sum To Product ..... 5

Centers of Triangles ..... 5

费马点 ..... 5

定义 ..... 5

作法 ..... 6

平面四边形费马点 ..... 6

Bretschneider's 公式 ..... 6

以六条棱表示的四面体 ..... 7

其他 ..... 7

基本代码 ..... 8

三维凸包 ..... 8

三维几何 ..... 13

两个三维球体积并 ..... 15

半平面交 ..... 15

圆和多边形面积交 ..... 20

绕轴旋转、三维空间点对直线垂线的垂足 ..... 22

凸包[Graham] – O(nlogn) ..... 23

最远点对 – O(n) ..... 24

最近点对问题 – 分治 – O(nlogn) ..... 24

周长最小三角形 – 分治 – O(nlogn) ..... 26

面积最大多边形 ..... 28

点集的最小覆盖圆 – 期望 O(n) ..... 31

外接圆最大三角形 – 枚举 1 点，极角排序 - O($n^2$logn)........................................32
给你一些线段,找到一条直线,要求穿过尽量多的线段 – O($n^2$logn)........................34
判断两个正交矩形是否有公共点.......................................................................35

# 缺：线段相交

# 几何公式

## 三角公式 Trigonometric Function Formula

### Product To Sum

$$\mathrm{sinAcosB} = \frac{\sin(A+B)+\sin(A-B)}{2}$$
$$\mathrm{cosAsinB} = \frac{\sin(A+B)-\sin(A-B)}{2}$$
$$\mathrm{sinAsinB} = -\frac{\cos(A+B)-\cos(A-B)}{2}$$
$$\mathrm{cosAcosB} = \frac{\cos(A+B)+\cos(A-B)}{2}$$

### Sum To Product

$$\mathrm{sinA}+\mathrm{sinB} = 2\sin\frac{A+B}{2}\cos\frac{A-B}{2}$$
$$\mathrm{sinA}-\mathrm{sinB} = 2\cos\frac{A+B}{2}\sin\frac{A-B}{2}$$
$$\mathrm{cosA}+\mathrm{cosB} = 2\cos\frac{A+B}{2}\cos\frac{A-B}{2}$$
$$\mathrm{cosA}-\mathrm{cosB} = -2\sin\frac{A+B}{2}\sin\frac{A-B}{2}$$

### Centers of Triangles

| | |
|---|---|
| **内心 Incenter**<br>$\mathrm{I} = \frac{\lvert\vec{a}\rvert\cdot A+\lvert\vec{b}\rvert\cdot B+\lvert\vec{c}\rvert\cdot C}{\lvert\vec{a}\rvert+\lvert\vec{b}\rvert+\lvert\vec{c}\rvert}$<br>**重心 Median**<br>$\mathrm{M} = \frac{A+B+C}{2}$ | **外心 Circumceter –** 详见点集的最小外接圆<br>(列方程求解，到所有点距离相等，正四面体外心同理)<br>**垂心**<br>$\mathrm{H} = \frac{\alpha A+\beta B+\gamma C}{\alpha+\beta+\gamma}$<br>$\alpha = (\vec{a}\cdot\vec{b})(\vec{a}\cdot\vec{c})$<br>$\beta = (\vec{b}\cdot\vec{c})(\vec{b}\cdot\vec{a})$<br>$\gamma = (\vec{c}\cdot\vec{a})(\vec{c}\cdot\vec{b})$ |

四面体块块 ABCD 的重心$\frac{A+B+C+\mathrm{D}}{2}$

## 费马点

### 定义

平面上有三个不在同一条直线上的点 *A*,*B*,*C*，对平面上的另一个点 *P*，考虑这一点到原来的三个点的距离之和：*PA* + *PB* + *PC*

费马点是这样一个点 $P_0$，使得它到点 *A*,*B*,*C* 的距离之和 $P_0A + P_0B + P_0C$ 比任何其它的

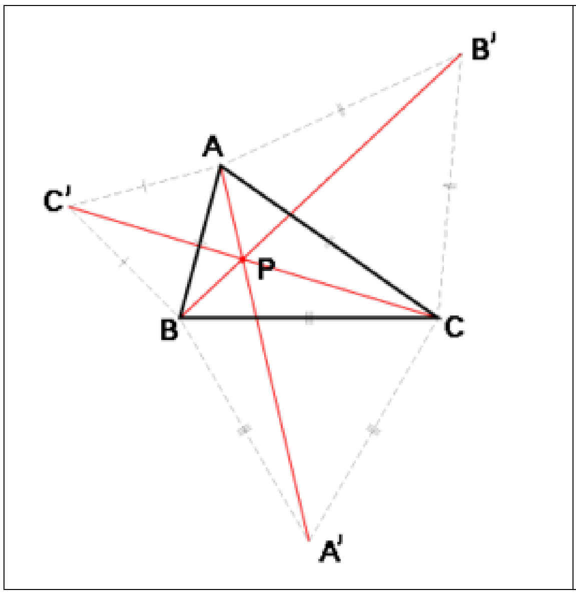
*PA* + *PB* + *PC* 都要小。

费马点的定义可以推广到更多点的情况。设平面上有 *m* 个点：$P_1$, $P_2$,..., $P_m$，又有正实数：$\lambda_1, \lambda_2, ..., \lambda_m$。费马问题可以推广为：寻找一个点 *X*，使得它到这 *m* 个点的距离之和：

$\lambda_1 XP_1 + \lambda_2 XP_2 + ... + \lambda_m XP_m$

是最小的。

## 作法

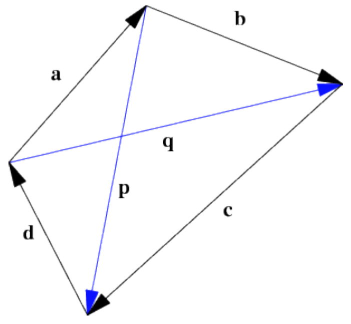
| | |
|---|---|
|  | 三角形的费马点的作法：<br>当有一个内角 ≥ 120 °时，费马点为此角对应顶点<br>当三角形的内角都 < 120 °时<br>以三角形的每一边为底边，向外做三个正三角形⊿*ABC'*，⊿*BCA'*，⊿*CAB'*<br>连接 *CC'*、*BB'*、*AA'*，则三条线段的交点就是所求的点 |

## 平面四边形费马点

1. 在凸四边形 ABCD 中，费马点 P 为两对角线 AC、BD 交点。
2. 在凹四边形 ABCD 中，费马点 P 为凹顶点。

四边形以上的多边形没有公式求费马点，因此可以使用随机化变步长贪心法

## Bretschneider's 公式



边长为 a, b, c, d 的面积为

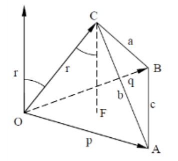
$$\mathrm{K} = \frac{1}{4}\sqrt{4p^2q^2 - (b^2 + d^2 - a^2 - c^2)^2} = \sqrt{(s-a)(s-b)(s-c)(s-d) - \frac{1}{4}(ac+bd+pq)(ac+bd-pq)}$$

$$\mathrm{K} = \sqrt{(s-a)(s-b)(s-c)(s-d) - abcd\cos^2\left[\frac{P+Q}{2}\right]}$$

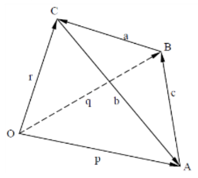
其中 p, q 为对角线长度，s 为半周长，P 和 Q 为四边形的一组对角。

ref http://mathworld.wolfram.com/BretschneidersFormula.html

## 以六条棱表示的四面体



$$288\mathrm{V}^2 = \begin{vmatrix} 2\mathrm{P} & \mathrm{P}+\mathrm{Q}-\mathrm{C} & \mathrm{P}+\mathrm{R}-\mathrm{B} \\ \mathrm{Q}+\mathrm{P}-\mathrm{C} & 2\mathrm{Q} & \mathrm{Q}+\mathrm{R}-\mathrm{A} \\ \mathrm{R}+\mathrm{P}-\mathrm{B} & \mathrm{R}+\mathrm{Q}-\mathrm{A} & 2\mathrm{R} \end{vmatrix} = \begin{vmatrix} 0 & \mathrm{P} & \mathrm{Q} & \mathrm{R} & 1 \\ \mathrm{P} & 0 & \mathrm{C} & \mathrm{B} & 1 \\ \mathrm{Q} & \mathrm{C} & 0 & \mathrm{A} & 1 \\ \mathrm{R} & \mathrm{B} & \mathrm{A} & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix}$$



其中 P, Q, R, A, B, C 分别表示 p, q, r, a, b, c 的平方

设四面体外接球半径为 R，则 6RV = j
其中 V 为四面体的体积，j 为以 e, f, g 为三边的三角形面积，e = ap, f = bq, g = cr
即：一个四面体的体积和其外接球半径的成绩的六倍等于三边各为四面体对棱乘积的三角形的面积。

设四面体内接球的半径 r，则 3V = rA
其中 V 为四面体的体积，A 为四面体的表面积。

## 其他

向量的混合积$\langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \rangle = (\boldsymbol{a} \times \boldsymbol{b}) \cdot \boldsymbol{c} = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}$，代表平行六面体的体积。

四维球的超体积公式：$\frac{1}{2}\pi^2 r^4$

四维球的表体积公式：$2\pi^2 r^3$

A、B、C 三点共线 *iff.* $\left|\overrightarrow{AB} \times \overrightarrow{AC}\right| < \varepsilon$

三角形内切圆半径$\mathrm{r} = \frac{2S}{a+b+c}$

三角形外接圆半径$\mathrm{R} = \frac{abc}{4S}$

**三角形**

中线长度：$\mathrm{Ma} = \frac{1}{2}\sqrt{2(b^2 + c^2) - a^2}$

内角平分线长度：$\mathrm{Ta} =$

## 基本代码

```
const double eps = 1e-8;
const double PI = acos(-1.0);
inline int sign(double a){ return a < -eps ? -1 : a > eps;}
inline int sign(int a){ return a < 0 ? -1 : a > 0;}
template <typename T> inline T sqr(T a){ return a * a;}
double ASIN(double x) {
	if ( fabs(x) > 1 ) x = sign(x); return asin(x);
}
double ACOS(double x) {
	if ( fabs(x) > 1 ) x = sign(x); return acos(x);
}
bool isinteger(double x){ return fabs(x - floor(x + eps)) < eps;}
```

## 三维凸包

```
const static double eps = 1e-8;

int sign(double x) {
	return x < -eps ? -1 : x > eps;
}

struct Point3D {
	double x, y, z;
	void read(){
		cin >> x >> y >> z;
	}
	Point3D(){
	}
	Point3D(double x, double y, double z) : x(x), y(y), z(z) {
	}
	Point3D operator + (const Point3D& o) const {
		return Point3D(x + o.x, y + o.y, z + o.z);
	}
	Point3D operator - (const Point3D& o) const {
		return Point3D(x - o.x, y - o.y, z - o.z);
	}
	Point3D operator * (double o) const {
		return Point3D(x * o, y * o, z * o);
	}
	Point3D operator / (double o) const {
		return Point3D(x / o, y / o, z / o);
	}
	Point3D operator ^ (const Point3D& o) const {
		return Point3D(y * o.z - z * o.y, z * o.x - x * o.z, x * o.y - y * o.x);
	}
	double operator * (const Point3D& o) const {
		return x * o.x + y * o.y + z * o.z;
	}
	friend Point3D cross(const Point3D& a, const Point3D& b, const Point3D& c) {
		return (b - a) ^ (c - a);
	}
	friend double mix(const Point3D& a, const Point3D& b, const Point3D& c) {
		return (a ^ b) * c;
	}
	friend bool isColinear(const Point3D& a, const Point3D& b, const Point3D& c) {
		return cross(a, b, c) == ZERO;
	}
	friend bool isCoplanar(const Point3D& a, const Point3D& b, const Point3D& c, const
Point3D& d) {
		return sign( mix(b - a, c - a, d - a) ) == 0;
	}
	friend bool pointInTri(const Point3D& o, const Point3D& a, const Point3D& b, const
Point3D& c) {
		Point3D x = cross(a, o, b), y = cross(b, o, c), z = cross(c, o, a);
		if ( sign( x * y ) > 0 && sign( y * z ) > 0 ) return true;
		if ( onSegment(o, a, b) || onSegment(o, b, c) || onSegment(o, c, a) ) return -1;
		return false;
	}
	friend int onSegment(const Point3D& v, const Point3D& a, const Point3D& b) {
		if ( !isColinear(v, a, b) ) return false;
		int flag = sign( (v - a) * (v - b) );
		if ( flag == 0 ) return -1;
		return flag < 0;
	}
	double length() const {
		return sqrt(x * x + y * y + z * z);
	}
	Point3D unit() const {
		return *this / length();
	}
	double project(const Point3D& o) const {
		return *this * o.unit();
	}
	bool operator < (const Point3D& o) const {
		if ( sign(x - o.x) != 0 ) return x < o.x;
		if ( sign(y - o.y) != 0 ) return y < o.y;
		if ( sign(z - o.z) != 0 ) return z < o.z;
		return false;
	}
	bool operator == (const Point3D& o) const {
		if ( sign(x - o.x) != 0 ) return false;
		if ( sign(y - o.y) != 0 ) return false;
		if ( sign(z - o.z) != 0 ) return false;
		return true;
	}
	static Point3D ZERO;
} Point3D::ZERO(0, 0, 0);

struct Poly3D {
	static const int maxn = 100 + 5;
	static const int maxf = maxn * maxn + 5;
	Point3D p[maxn]; int n;
	int f[maxf][3]; int fc;

	void read(int n){
		for (int i = 0; i < n; ++i) p[i].read();
		sort(p, p + n); n = unique(p, p + n) - p;
		this->n = n; fc = 0;
	}

	bool convex(){ // cross(f[i][0], f[i][1], f[i][2]) point to outer
		random_shuffle(p, p + n);
		if ( !findTet() ) return false;
		for (int i = 3; i < n; ++i) {
			addpoint(i);
		}
		return true;
	}

	bool findTet() {
		for (int i = 2; i < n; ++i) {
			if ( !isColinear(p[0], p[1], p[i]) ) {
				swap(p[2], p[i]);
```

```
                for (int j = i + 1; j < n; ++j) {
                    if ( !isCoplanar(p[0], p[1], p[2], p[j]) ) {
                        swap(p[3], p[j]);
                        addface(0, 1, 2); addface(0, 2, 1);
                        return true;
                    }
                }
                return false;
            }
        }
        return false;
    }

    void addpoint(int v) {
        static int mark[maxn][maxn], cnt = 0;
        ++cnt;
        bool flag = false;
        for (int i = 0; i < fc; ) {
            int a = f[i][0], b = f[i][1], c = f[i][2];
            if ( sign( mix(p[a] - p[v], p[b] - p[v], p[c] - p[v]) ) < 0 ) {
                flag = true;
                mark[a][b] = mark[b][a] = mark[a][c] = mark[c][a]
                    = mark[b][c] = mark[c][b] = cnt;
                delface(i);
            } else {
                ++i;
            }
        }
        if ( !flag ) return;

        int _fc = fc;
        for (int i = 0; i < _fc; ++i) {
            int a = f[i][0], b = f[i][1], c = f[i][2];
            if ( mark[a][b] == cnt ) addface(b, a, v);
            if ( mark[b][c] == cnt ) addface(c, b, v);
            if ( mark[c][a] == cnt ) addface(a, c, v);
        }
    }

    void addface(int a, int b, int c) {
        f[fc][0] = a, f[fc][1] = b, f[fc][2] = c; ++fc;
    }

    void delface(int i) {
        memmove(f[i], f[--fc], sizeof(f[i]) );
}

int in(const Point3D& o) const { /* -1 on face */
        for (int i = 0; i < fc; ++i) {
                const Point3D &a = p[ f[i][0] ], b = p[ f[i][1] ], c = p[ f[i][2] ];
                int flag = sign( mix(a - o, b - o, c - o) );
                if ( flag == 0 && pointInTri(o, a, b, c) ) return -1;
                if ( flag < 0 ) return false;
        }
        return true;
}

double dist(const Point3D& o) const {
        double ans = inf;
        for (int i = 0; i < fc; ++i) {
                const Point3D &a = p[ f[i][0] ], b = p[ f[i][1] ], c = p[ f[i][2] ];
                Point3D normal = (b - a) ^ (c - a);
                double d = (a - o).project(normal);
                checkmin(ans, d);
        }
        return ans;
}

int facecnt() const {
        static Point3D normal[maxf];
        for (int i = 0; i < fc; ++i) {
                const Point3D &a = p[ f[i][0] ], b = p[ f[i][1] ], c = p[ f[i][2] ];
                normal[i] = cross(a, b, c).unit();
        }
        sort(normal, normal + fc);
        return unique(normal, normal + fc) - normal;
}

double surface() const {
        double ans = 0;
        for (int i = 0; i < fc; ++i) {
                const Point3D &a = p[ f[i][0] ], b = p[ f[i][1] ], c = p[ f[i][2] ];
                ans += cross(a, b, c).length();
        }
        return ans / 2;
}

double volume() const {
		double ans = 0;
		Point3D o = p[0];
		for (int i = 0; i < fc; ++i) {
			const Point3D &a = p[ f[i][0] ], b = p[ f[i][1] ], c = p[ f[i][2] ];
			ans += mix(a - o, b - o, c - o);
		}
		return ans / 6;
	}
	Point3D massCenter() const {
		Point3D ans = Point3D::ZERO;
		double vol = 0;
		const Point3D o = p[0];
		for (int i = 0; i < fc; ++i) {
			const Point3D &a = p[ f[i][0] ], b = p[ f[i][1] ], c = p[ f[i][2] ];
			double v = mix(a - o, b - o, c - o);
			ans = ans + (o + a + b + c) * (v / 4);
			vol += v;
		}
		return ans / vol;
	}
} poly;

void solve(int n) {
	poly.read(n); poly.convex(); Point3D o = poly.massCenter();
	printf("%.3f\n", poly.dist(o) );
}

int main(){ for (int n; cin >> n; solve(n)); }
```

# 三维几何

```
struct LineAV3D {
	Point3D a, v;
	#define LineST3D( s, t ) LineAV3D( (s), (t) - (s) )
	LineAV3D(){
	}
	LineAV3D(const Point3D& a, const Point3D& v) : a(a), v(v) {
	}
};

struct Plane3D {
	Point3D a, b, c;
	Point3D normal() const {
        return cross(a, b, c);
    }
};

struct Geom3D {
    double dist(const Point3D& a, const Point3D& b)    {
        return (b - a).length();
    }
    double dist(const Point3D& p, const LineAV3D& ln)    {
        double area2 = ( (p - ln.a) ^ ln.v ).length();
        return area2 / ln.v.length();
    }
    double dist(const Point3D& p, const Plane3D& s)    {
        return (p - s.a).project( s.normal() );
    }
    double dist(const LineAV3D& x, const LineAV3D& y)    {
        Point3D n = x.v ^ y.v;
        if ( n.isZero() ) return dist(x.a, y); else return (x.a - y.a).project(n);
    }
    Point3D foot(const Point3D &p, const LineAV3D &ln)    {
        return ln.a + ln.v.unit() * (p - ln.a).project(ln.v);
    }
    Point3D foot(const Point3D &p, const Plane3D& s)    {
        Point3D n = s.normal();
        return p + n.unit() * (s.a - p).project(n);
    }
    int intersect(const LineAV3D& ln, const Plane3D& s, Point3D &p)    {
        Point3D n = s.normal();
        double x = ln.v.project(n);
        if ( sign(x) == 0 ) {
            if ( sign( dist(ln.a, s) ) == 0 ) return -1; /* infinity */
            else return 0; /* parallel */
        }
        double t = ( foot(ln.a, s) - ln.a ).length() / x;
        p = ln.a + ln.v * t;
        return 1;
    }
    int intersect(const Plane3D& x, const Plane3D& y, LineAV3D& ln)    {
        Point3D nx = x.normal(), ny = y.normal();
        Point3D v = nx ^ ny;
        if ( v.isZero() ) return 0;
        Point3D a; intersect( LineST3D(x.a, x.b), y, a);
        ln = LineAV3D(a, v); return 1;
    }
```

```
	double angle(const Point3D& x, const Point3D& y) { //vector
		double cos = x * y / ( x.length() * y.length() );
		return acos( fabs(cos) );
	}
	double angle(const LineAV3D& x, const LineAV3D& y) {
		return angle(x.v, y.v);
	}
	double angle(const Plane3D& x, const Plane3D& y) {
		return angle( x.normal(), y.normal() );
	}
} geom;
```

## 两个三维球体积并

```
double integral(double r, double z) {
	return PI * (r * r - z * z / 3) * z;
}

struct Sphere {
	Point3D c; double r;
};

double unionArea(const Sphere &a, const Sphere &b) {
	double d = (a.c - b.c).length(), r1 = a.r, r2 = b.r;
	if ( r1 < r2 ) swap(r1, r2);
	double t = (r1 * r1 - r2 * r2) / d, h1 = (d + t) / 2, h2 = (d - t) / 2;
	if ( sign( r1 - r2 - d) >= 0 ) h1 = r1, h2 = -r2;
	if ( sign(r1 + r2 - d) <= 0 ) h1 = r1, h2 = r2;
	return integral(r1, h1) - integral(r1, -r1) + integral(r2, h2) - integral(r2, -r2);
}
```

# 半平面交

```
//已经通过 5 题，上次使用 2012.7.31
#define rep(i, n) for (int i = 0; i < (n); ++i)
const double eps = 1e-8;
int sign(double x){ return x < -eps ? -1 : x > eps; }

struct Point {
	double x, y;
	void read(){ scanf("%lf%lf", &x, &y); }
	Point(){}
	Point(double x, double y) : x(x), y(y) {}
	Point operator + (const Point &o) const { return Point(x + o.x, y + o.y); }
	Point operator - (const Point &o) const { return Point(x - o.x, y - o.y); }
	Point operator * (double o) const { return Point(x * o, y * o); }
	Point operator / (double o) const { return Point(x / o, y / o); }
	double operator * (const Point &o) const { return x * o.x + y * o.y; }
	double operator ^ (const Point &o) const { return x * o.y - y * o.x; }
	bool operator == (const Point &o) const {
		return !sign(x - o.x) && !sign(y - o.y);
	}
	int quad() const { return sign(x) >= 0 ? sign(y) >= 0 ? 1 : 4 : sign(y) >= 0 ? 2 : 3; }
	double length() const { return sqrt( x * x + y * y ); }
	Point setLength(double d) const { return *this * (d / length()); }
	Point unit() const { return *this / length(); }
	double project(const Point &n) const { //投影到 n 上的长度
		return *this * n.unit();
	}
	friend int intersect(const Point& a, const Point& v, const Point& b, const Point& u, Point&p){
		// a + v[t] = b + u[s] => v[t] - u[s] = b - a = c
		Point c = b - a; double d = u ^ v;
		if ( sign(d) == 0 ) { /* assume v != 0 && u != 0 */
			if ( sign(c ^ u) == 0 ) return -1; /* coincide */
			return 0; /* parallel */
		}
		double t = (u ^ c) / d; p = a + v * t; return 1; /* intersect */
	}
};

struct LineAV {
	Point a, v;
	#define LineST(a, b) LineAV( (a), (b) - (a) )
	void read(){ Point a, b; a.read(); b.read(); *this = LineST(a, b); }
	LineAV(){}
	LineAV(const Point &a, const Point &v) : a(a), v(v) {}
	LineAV offset(double d) const {
		return LineAV( a + Point(-v.y, v.x).setLength(d), v );
	}
	bool operator < (const LineAV &o) const {
		int dq = v.quad() - o.v.quad(); if (dq != 0) return dq < 0;
		int x = sign( v ^ o.v ); if (x != 0) return x > 0;
		return ( (o.a - a) ^ v ) > 0;
	}
	bool operator == (const LineAV &o) const {
		int dq = v.quad() - o.v.quad(); if (dq != 0) return false;
```

```
        int x = sign( v ^ o.v ); if (x != 0) return false;
        return true;
    }
    friend int intersect(const LineAV &x, const LineAV& y, Point &p) {
        return intersect(x.a, x.v, y.a, y.v, p);
    }
};

struct Geom {
    double cross(const Point &a, const Point& b, const Point &c) {
        // cross(a, b, c) > 0 iff. c left of ray a->b
        return (b - a) ^ (c - a);
    }
    bool onSegment(const Point &p, const Point &a, const Point &b) {
        if ( cross(a, b, p) != 0 ) return 0;
        return between(p.x, a.x, b.x) && between(p.y, a.y, b.y);
    }
    bool between(double t, double x, double y){
        if (x > y) swap(x, y);
        return sign(x-t) <= 0 && sign(t-y) <= 0;
    }
} geom;

struct HalfPlane {
    bool out(const LineAV& x, const LineAV& y, const LineAV& z) {
        Point p; intersect(x, y, p);
        int d = sign( z.v ^ (p - z.a) ); if ( d != 0 ) return d < 0;
        int t = sign( x.v ^ z.v ) * sign( x.v ^ y.v ); return t > 0;
    }
    void solve(LineAV ls[], int &n, int &s, int &t) {
        sort(ls, ls + n); n = unique(ls, ls + n) - ls;
        int i, j;
        for (s = 0, t = 1, i = 2; i < n; ++i) {
            while ( s < t && out(ls[t-1], ls[t], ls[i]) ) --t;
            while ( s < t && out(ls[s+1], ls[s], ls[i]) ) ++s;
            ls[++t] = ls[i];
        }
        do {
            n = t - s + 1;
            while ( s < t && out(ls[t-1], ls[t], ls[s]) ) --t;
            while ( s < t && out(ls[s+1], ls[s], ls[t]) ) ++s;
        } while ( n != t - s + 1 );
        ls[t+1] = ls[s];
    }
```

```
} halfPlane;

Point vertex[] = { //千万要逆时针给出
        Point(-inf, -inf), Point(inf, -inf), Point(inf, inf), Point(-inf, inf),
};
LineAV edges[] = {
        LineST( vertex[0], vertex[1] ), LineST( vertex[1], vertex[2] ),
        LineST( vertex[2], vertex[3] ), LineST( vertex[3], vertex[4] ),
};

struct Poly {
        Point p[maxn]; int n;
        bool read(){
                if( !(cin >> n) || n == 0 ) return false;
                rep(i, n) p[i].read(); p[n] = p[0];
                if ( area() < 0 ) reverse(p + 1, p + n); return true;
        }
        double area() const {
                double area = 0; rep(i, n) area += p[i] ^ p[i+1]; return area / 2;
        }
        Point centroid() const {
                Point ans = p[0]; for (int i = 1; i < n; ++i) ans = ans + p[i]; return ans / n;
        }
        int in(const Point &o) const{
                /* 1 -> strict in, 0 -> strict out, -1 -> onEdge, -2 -> onVertex*/
                rep(i, n) if (p[i] == o) return -2;
                rep(i, n) if ( geom.onSegment(o, p[i], p[i+1]) ) return -1;
                int wn = 0;
                rep(i, n) {
                        int k = sign( geom.cross(p[i], p[i+1], o) );
                        int d1 = sign( p[i].y - o.y ), d2 = sign( p[i+1].y - o.y );
                        if ( k > 0 && d1 <= 0 && d2 > 0 ) ++wn;
                        if ( k < 0 && d2 <= 0 && d1 > 0 ) --wn;
                }
                return wn != 0;
        }
        bool calcCore(Poly &ans /* 可以为*this */, double d = 0 /*offset*/) const {
                static LineAV ls[maxn];
                if ( area() > 0 ) {
                        rep(i, n) ls[i] = LineST(p[i], p[i+1]).offset(d);
                } else {
                        rep(i, n) ls[i] = LineST(p[i+1], p[i]).offset(d);
                }
                rep(i, 4) ls[i+n] = edges[i]; ans.n = n + 4;
        int s, t; halfPlane.solve(ls, ans.n, s, t);
        if ( ans.n < 3 ) return false;
        for (int i = s; i <= t; ++i) intersect(ls[i], ls[i+1], ans.p[i-s]);
        ans.p[ ans.n ] = ans.p[0]; return true;
    }
    void largestCircle(Point &o, double &r) const { //最大内切圆
        double low = 0, high = inf, mid;
        static Poly poly;
        while ( low < high - eps ) {
            mid = (low + high) / 2;
            if ( calcCore(poly, mid) ) {
                r = low = mid;
            } else {
                high = mid;
            }
        }
        calcCore(poly, r); o = poly.centroid();
    }
    pair<Point, Point> mostDistance() const { //最远点对
        double r = -1, t;
        Point x, y;
        for (int i = 0, j = 1; i < n; ++i) {
            while ( cross(p[i], p[i+1], p[j+1]) > cross(p[i], p[i+1], p[j]) + eps )
                if (++j == n) j = 0;
            if ( ( t = (p[i] - p[j]).length() ) > r ) {
                r = t; x = p[i]; y = p[j];
            }
            if ( ( t = (p[i+1] - p[j]).length() ) > r ) {
                r = t; x = p[i+1]; y = p[j];
            }
        }
        return make_pair(x, y);
    }
} poly;

struct Convex : Poly {
    int _in(const Point &o, int i) const {
        if ( o == p[i] || o == p[i+1] ) return -2;
        if ( geom.onSegment(o, p[i], p[i+1]) ) return -1;
        return sign( geom.cross(p[i], p[i+1], o) ) > 0 ? 1 : 0;
    }
    int in(const Point &o) const{ //REQUIRE: no point colinear
        /* 1 -> strict in, 0 -> strict out, -1 -> onEdge, -2 -> onVertex*/
        if ( _in(o, 0) != 1 ) return _in(o, 0);
```

```
		if ( _in(o, n-1) != 1 ) return _in(o, n-1);
		int low = 1, high = n - 2, k;
		while ( low <= high ) {
			int mid = (low + high) / 2;
			if ( sign( geom.cross(p[0], p[mid], o) ) >= 0 ) {
				k = mid; low = mid + 1;
			} else {
				high = mid - 1;
			}
		}
		if ( o == p[k] || o == p[k+1] ) return -2;
		int s = sign( geom.cross(p[k], p[k+1], o) );
		return s == 0 ? -1 : s > 0;
	}
	int intersect(const LineAV &ln, Point p[]) const { //TODO
	}
} convex;
```

## 圆和多边形面积交

```
struct Circle {
	Point o; double r;
	double area(double theta = 2 * PI) const {
		return theta * r * r * 0.5;
	}
	int intersect(Point a, const Point& v, double &t1, double &t2) const { //圆，线段求交
		a = a - o;
		double A = v * v, B = a * v * 2, C = a * a - r * r;
		double D = B * B - 4 * A * C;
		t1 = t2 = NAN;
		switch( sign(D) ) {
			case 1:
				D = sqrtl(D);
				t1 = ( -B - (sign(B) >= 0 ? 1 : -1) * D ) / ( 2 * A ), t2 = C / (A * t1);
				break;
			case 0:
				t1 = t2 = -B / (2 * A);
				break;
		}
		if ( t1 > t2 ) swap(t1, t2);
		if ( isnan(t2) || !( sign(t2) >= 0 && sign(t2 - 1) <= 0) ) { t2 = NAN; }
		if ( isnan(t1) || !( sign(t1) >= 0 && sign(t1 - 1) <= 0) ) { t1 = t2; t2 = NAN; }
		return !isnan(t1) + !isnan(t2) - (t1 == t2);
```

```
        }
};

struct Geom {
        double angle(const Point& a, const Point& b) { //向量夹角[0, PI)
                return ACOS( a * b / ( a.length() * b.length() ) );
        }
} geom;

struct CirclePolyIntersectArea {
        double gao(const Circle &c, Point a, Point b) {
                double la = a * a, lb = b * b, lr = c.r * c.r, ans = sign(a ^ b);
                if ( !sign(la) || !sign(lb) || !ans ) return 0; //一定要加
                if ( la > lb ) { swap(a, b); swap(la, lb); }
                if ( sign(lb - lr) <= 0 ) {
                        ans *= fabs(a ^ b) * 0.5;
                } else {
                        double t1, t2; Point v = b - a;
                        c.intersect(a, v, t1, t2);
                        if ( sign(la - lr) < 0 ) {
                                assert( !isnan(t1) && isnan(t2) );
                                Point p = a + v * t1;
                                ans *= fabs(a ^ p) * 0.5 + c.area( geom.angle(p, b) );
                        } else {
                                if ( isnan(t1) ) {
                                        ans *= c.area( geom.angle(a, b) );
                                } else {
                                        if ( isnan(t2) ) t2 = t1;
                                        Point p = a + v * t1, q = a + v * t2;
                                        ans *= fabs(p ^ q) * 0.5 + c.area( geom.angle(a, p) + geom.angle(q, b) );
                                }
                        }
                }
                return ans;
        }
        double solve(const Circle &c, Point p[], int n){
                static Point q[maxn]; double ans = 0;
                rep(i, n) q[i] = p[i] - c.o; q[n] = q[0];
                rep(i, n) ans += gao(c, q[i], q[i+1]);
                return fabs(ans);
        }
        double solveCircleTri(Circle c, Point x, Point y, Point z) {
                Point o = c.o; x = x - o; y = y - o; z = z - o; c.o = Point(0, 0);
                return fabs(gao(c, x, y) + gao(c, y, z) + gao(c, z, x) );
```

```
    }} task;

int main(){ //poj3675
    for (double r; cin >> r; ) {
        int n; cin >> n;
        static Point p[maxn]; rep(i, n) p[i].read();
        Circle c; c.o = Point(0, 0); c.r = r;
        printf( "%.2f\n", task.solve(c, p, n) );
    }
}
```

# 绕轴旋转、三维空间点对直线垂线的垂足

设 P(x,y,z)，绕单位向量$u = (u_x, u_y, u_z)$ $(u_x^2 + u_y^2 + u_z^2 = 1)$旋转θ后的坐标为

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \mathrm{R} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

其中

$$\begin{aligned}\mathrm{R} &= \begin{pmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) - u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{pmatrix} \\ &= \mathrm{I}\cos\theta + \sin\theta[\boldsymbol{u}]_\times + (1-\cos\theta)\boldsymbol{u} \otimes \boldsymbol{u}\end{aligned}$$

其中

$$\boldsymbol{u} \otimes \boldsymbol{u} = \begin{pmatrix} u_x^2 & u_x u_y & u_x u_z \\ u_y u_x & u_y^2 & u_y u_z \\ u_z u_x & u_z u_y & u_z^2 \end{pmatrix}, [\boldsymbol{u}]_\times = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}$$

此外，对于 P 绕 AB 旋转的情况(设$\mathbf{u} = \mathbf{AB}/|\mathbf{A}\mathrm{B}|$)，则旋转矩阵为

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} & & & x_A \\ & R & & y_A \\ & & & z_A \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -x_A \\ 0 & 1 & 0 & -y_A \\ 0 & 0 & 1 & -z_A \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

```
//已经通过 hdu2014, 1006, brbin regional online 2010
#define sqr(x) ((x)*(x))
struct Point{
    double x, y, z;
    Point(double a, double b, double c){ x = a; y = b; z = c; }
    Point() {}
};

Point rotate(double angle, Point a, Point b, Point p){ //点 P 绕轴AB旋转(右手系)
    double ab2 = sqr(a.x-b.x) + sqr(a.y-b.y) + sqr(a.z-b.z);
    double lamda = ((b.x-a.x)*(b.x-p.x) + (b.y-a.y)*(b.y-p.y) + (b.z-a.z)*(b.z-p.z)) / ab2;
    Point q = Point(
        a.x * lamda + b.x * (1-lamda),
        a.y * lamda + b.y * (1-lamda),
        a.z * lamda + b.z * (1-lamda)
    );   //q 为 p 在 ab 上的垂足
    double sinadivab = sin(angle) / sqrt(ab2);
    double cosa = cos(angle);
    return Point(
        (p.x-q.x)*cosa + q.x + ((b.y-a.y)*(p.z-q.z)-(b.z-a.z)*(p.y-q.y)) * sinadivab,
        (p.y-q.y)*cosa + q.y + ((b.z-a.z)*(p.x-q.x)-(b.x-a.x)*(p.z-q.z)) * sinadivab,
        (p.z-q.z)*cosa + q.z + ((b.x-a.x)*(p.y-q.y)-(b.y-a.y)*(p.x-q.x)) * sinadivab
    );
}

Point foot(Point a, Point b, Point p){ //返回 p 在 ab 上的垂足
    double ab2 = sqr(a.x-b.x) + sqr(a.y-b.y) + sqr(a.z-b.z);
    double lamda = ((b.x-a.x)*(b.x-p.x) + (b.y-a.y)*(b.y-p.y) + (b.z-a.z)*(b.z-p.z)) / ab2;
    return Point(
        a.x * lamda + b.x * (1-lamda),
        a.y * lamda + b.y * (1-lamda),
        a.z * lamda + b.z * (1-lamda)
    );
}
```

## 凸包[Graham] – O(nlogn)

```
//已经通过 4 题，上次使用 2012-6-28
double cross(const Point& a, const Point &b, const Point &c){ //ab x ac
    return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}

//水平序，分左右链，返回凸包点数 k，凸包按逆时针顺序存在 q[]中，q[k] = q[0]
//要包含共线点的话，一定要提前 unique 一下
int graham(Point p[], int n, Point q[]){
    int i, kk, k;
    sort( p, p + n, lessx() );
    if (1 == n){ q[0] = q[1] = p[0]; return 1;}
    for (k = 0, i = 0; i < n; q[k++] = p[i++])
        while (k >= 2 && cross(q[k-2], q[k-1], p[i]) <= eps) //要包含共线点则  < -eps
            --k;
    for (kk = k, i = n-2; i >= 0; q[k++] = p[i--])
        while (k > kk && cross(q[k-2], q[k-1], p[i]) <= eps) //要包含共线点则  < -eps
            --k;
    return k-1;
}
```

## 最远点对 – O(n)

//已经通过 4 题，上次使用 2012-8-2
证明：最远点对 -> 同旁内角和小于 180° -> 对踵点
对于要距离最近的平行线把凸包夹住，则改为取最小值即可
凸包 p[]中的最远点对，要求 p[n] = p[0]，代码参考 P19 的 mostDistance 函数

## 最近点对问题 – 分治 – O(nlogn)

**//已经通过 hdu1007, cugb1251,**

```
const int maxn = 100005;

struct Point{
	double x, y; int index;
	friend bool lessx(const Point& a, const Point& b){ return a.x < b.x;}
	friend bool lessy(const Point& a, const Point& b){ return a.y < b.y;}
};

double dis2(Point p, Point q){ return sqr(p.x - q.x) + sqr(p.y - q.y); }

void partition(Point c[], Point a[], Point b[], int left, int mid, int right){
	//将 b[left..right]分成 c[left..mid]和 c[mid+1..right]
	//使得集合{a[p..m]} = {c[p..m]}, {a[m+1..q]} = {c[m+1..q]},且 c 是对 y 有序的
	//初始条件：b[left..right]是对 y 有序的，且集合{a[left..right]} = {b[left..right]}
	int j = left, k = mid+1;
	for (int i = left; i <= right; i++)
		if (b[i].index <= mid)
			c[j++] = b[i]; //数组 c 左半部保存划分后左部的点，且对 y 是有序的.
		else
			c[k++] = b[i];
}

void merge(Point p[], Point q[], int left, int mid, int right){
	//q[left..mid], q[mid+1..right]分别是对 y 有序的，将其合并储存到 p[left..right]
	int i = left, j = mid + 1, k = left;
	while (i <= mid && j <= right)
		if (q[i].y <= q[j].y) p[k++] = q[i++]; else p[k++] = q[j++];
	while (i <= mid) p[k++] = q[i++];
	while (j <= right) p[k++] = q[j++];
```

```
}

double closest2(Point a[], Point b[], Point c[], int left, int right){
	//求 a[p..q]的最近点对距离的平方
	//初始条件：集合{b[left..right]}={a[left..right]}，且 a 对 x 有序, b 对 y 有序
	if (right - left == 1)
		return dis2(a[left], a[right]);
	if (right - left == 2){
		double x1 = dis2(a[left], a[right]);
		double x2 = dis2(a[left + 1], a[right]);
		double x3 = dis2(a[left], a[left + 1]);
		return min(min(x1, x2), x3);
	}

	int mid = (left + right) / 2;
	partition(c, a, b, left, mid, right);
	double d1 = closest2(a, c, b, left, mid);
	double d2 = closest2(a, c, b, mid + 1, right);
	double dm = min(d1, d2);
	merge(b, c, left, mid, right);

	//找出离划分基准左右不超过 dm 的部分,存到 c[left..k-1], 且仍然对 y 坐标有序.
	int k = left;
	for (int i = left; i <= right; i++)
		if (sqr(b[i].x - b[mid].x) < dm) c[k++] = b[i];

	//用 c[p..k-1]中的点更新 dm
	for (int i = left; i < k; i++)
		for (int j = i + 1; j < k && sqr(c[j].y - c[i].y) < dm; j++)
			dm = min(dm, dis2(c[i], c[j]));
	return dm;
}

Point a[maxn], b[maxn], c[maxn];

int main(){
	int n, i; double d2;
	while(cin >> n && n){
		for (i = 0; i < n; i++)
			scanf("%lf%lf", &a[i].x, &a[i].y);
		sort(a, a + n, lessx);
		for (i = 0; i < n; i++) a[i].index = i;
		copy(a, a + n, b);
		sort(b, b + n, lessy);
		d2 = closest2(a, b, c, 0, n - 1);
		printf("%.2f\n", sqrt(d2) * 0.5);
	}
	return 0;
}
```

# 周长最小三角形 – 分治 – O(nlogn)

```
//已经通过 hdu3548
//基本 cp 某人标程 http://code.google.com/codejam/contest/dashboard?c=311101#s=a&a=1

template<class T> inline int size(const T&c) { return c.size(); }

const int BILLION = 1000000000;
const double INF = 1e20;
const double EPS = 1e-11;
typedef long long LL;

struct Point {
  int x,y;
  Point() {}
  Point(int x,int y):x(x),y(y) {}
};

inline Point middle(const Point &a, const Point &b) {
	return Point((a.x + b.x) / 2, (a.y + b.y) / 2);
}

struct CmpX {
	inline bool operator()(const Point &a, const Point &b) {
		if(a.x != b.x) return a.x < b.x;
		return a.y < b.y;
	}
} cmpx;

struct CmpY {
	inline bool operator()(const Point &a, const Point &b) {
		if(a.y != b.y) return a.y < b.y;
		return a.x < b.x;
	}
} cmpy;

inline LL sqr(int x) { return LL(x) * LL(x); }
inline double dist(const Point &a, const Point &b) {
	return sqrt(double(sqr(a.x - b.x) + sqr(a.y - b.y)));
}

inline double perimeter(const Point &a, const Point &b, const Point &c) {
	double x = dist(a,b), y = dist(b,c), z = dist(c,a);
	if (fabs(x + y - z) < EPS || fabs(x - y) > z - EPS) return INF;
	return x + y + z;
}

double calc(int n, const Point points[], const vector<Point> &pointsByY) {
	if(n < 3) return INF;

	int left = n / 2;
	int right = n - left;
	Point split = middle(points[left-1], points[left]);

	vector<Point> pointsByYLeft, pointsByYRight;
	pointsByYLeft.reserve(left);
	pointsByYRight.reserve(right);

	for (int i = 0; i < n; i++) {
		if(cmpx(pointsByY[i], split))
			pointsByYLeft.push_back(pointsByY[i]);
		else
			pointsByYRight.push_back(pointsByY[i]);
	}

	double res = INF;
	res = min(res, calc(left, points, pointsByYLeft));
	res = min(res, calc(right, points + left, pointsByYRight));

	static vector<Point> closeToTheLine;
	int margin = (res > INF / 2) ? BILLION : int(res / 2);
	closeToTheLine.clear();
	closeToTheLine.reserve(n);

	int start = 0;
	for(int i = 0;i < n; ++i) {
		Point p = pointsByY[i];
		if(abs(p.x - split.x) > margin) continue;
		while(start < size(closeToTheLine) && p.y - closeToTheLine[start].y > margin) ++start;
		for(int i = start; i < size(closeToTheLine); ++i) {
```

```
            for(int j = i+1; j < size(closeToTheLine); ++j) {
                res = min(res, perimeter(p, closeToTheLine[i], closeToTheLine[j]));
            }
        }
        closeToTheLine.push_back(p);
    }
    return res;
}

double calc(vector<Point> &points) {
    sort(points.begin(), points.end(), cmpx);
    vector<Point> pointsByY = points;
    sort(pointsByY.begin(), pointsByY.end(), cmpy);
    return calc(size(points), &points[0], pointsByY);
}

int main() {
    int t, n;
    scanf("%d", &t);
    for (int icase = 1; icase <= t; icase++){
        scanf("%d", &n);
        vector<Point> points; points.reserve(n);
        for (int i = 0; i < n; i++){
            int x, y;
            scanf("%d%d", &x, &y);
            points.push_back(Point(2 * x, 2 * y));
        }

        printf("Case %d: ", icase);
        double res = calc(points);

        if (res >= INF - EPS){
            printf("No Solution\n");
        }
        else {
            printf("%.3f\n", res / 2);
        }
    }
}
```

# 面积最大多边形

方法正确性详见 IEEE 论文 On a general method for maximizing and minimizing among certain geometric problems (由于论文思维跳跃过大，笔者没看懂 TAT)

```
//要求 p[1..n-1]是凸包且 p[n] = p[0]，时间复杂度 O(n)
double max_area_triangle(point p[], int n){ //hdu2202
    if (n < 3) return 0;
    int a = 0, b = 1, c = 2, ra = a, rb = b, rc = c;
    double A = area(p[a], p[b], p[c]);
    #define inc(x) do { if (++x == n) x = 0; } while (0)
    for (;;){
        for(;;) {
            while ( area(p[a], p[b], p[c+1]) >= area(p[a], p[b], p[c]) ) inc(c);
            if ( area(p[a], p[b+1], p[c]) >= area(p[a], p[b], p[c]) ) inc(b); else break;
        }
        if ( area(p[a], p[b], p[c]) > A ){
            ra = a, rb = b, rc = c; A = area(p[a], p[b], p[c]);
        }
        inc(a);
        if (a == b) inc(b);
        if (b == c) inc(c);
        if (a == 0) break;
    }
    return A;
    #undef inc
}

double max_area_quadrangle(point p[], int n){ //bzoj1069
    if (n == 3) return area(p[0], p[1], p[2]);
    if (n < 4) return 0;
    int a = 0, b = 1, c = 2, d = 3, ra = a, rb = b, rc = c, rd = d;
    double A = area(p[a], p[b], p[c], p[d]);
    #define inc(x) do { if (++x == n) x = 0; } while (0)
    for (;;){
        for (;;){
            for(;;) {
                while ( area(p[c], p[d+1], p[a]) >= area(p[c], p[d], p[a]) ) inc(d);
                if ( area(p[b], p[c+1], p[d]) >= area(p[b], p[c], p[d]) ) inc(c); else break;
            }
            if ( area(p[a], p[b+1], p[c]) >= area(p[a], p[b], p[c]) ) inc(b); else break;
        }
        if ( area(p[a], p[b], p[c], p[d]) > A ){
            ra = a, rb = b, rc = c, rd = d; A = area(p[a], p[b], p[c], p[d]);
        }
        inc(a);
        if (a == b) inc(b);
        if (b == c) inc(c);
        if (c == d) inc(d);
        if (a == 0) break;
    }
    return A;
    #undef inc
}

void dfs(int dep, point p[], int n, int k, int idx[], int opt[]){
    #define inc(x) do { if (++x == n) x = 0; } while (0)
    if (dep == k - 1) {
        while( area(p[idx[k-2]],p[idx[k-1]+1],p[idx[0]])>=area(p[idx[k-2]],p[idx[k-1]],p[idx[0]]) )
            inc(idx[k-1]);
        return;
    }
    dfs(dep+1, p, n, k, idx, opt);
    while( area( p[idx[dep-1]], p[idx[dep]+1], p[idx[dep+1]])
            >= area( p[idx[dep-1]], p[idx[dep]], p[idx[dep+1]]) ) {
       inc( idx[dep] );
       dfs(dep+1, p, n, k, idx, opt);
    }
}

double max_area_kgon(point p[], int n, int k){ //bzoj 1069
    if (n < 3) return 0;
    static int maxk = 10;
    double A = 0;
    for (int i = 3; i < k; i++){
        A += area(p[0], p[i-2], p[i-1]);
        if ( n == i ) return A;
    }
    A += area(p[0], p[k-2], p[k-1]);
    int idx[maxk], opt[maxk];
    for (int i = 0; i < k; i++)
        idx[i] = opt[i] = i;
    for (;;){
        dfs(1, p, n, k, idx, opt);
        double A0 = 0;
        for (int i = 3; i <= k; i++)
            A0 += area(p[idx[0]], p[idx[i-2]], p[idx[i-1]]);
        if ( A0 > A ){
            for (int i = 0; i < k; i++) opt[i] = idx[i];
            A = A0;
        }
        inc(idx[0]);
        for (int i = 1; i < k; i++)
            if (idx[i] == idx[i-1] ) inc(idx[i]);
        if (idx[0] == 0) break;
    }
    return A;
    #undef inc
}
```

# 点集的最小覆盖圆 – 期望 O(n)

```
//已经通过 zju1450
struct Point{ double x, y;};
struct Circle{ Point c; double r;};

double Distance(const Point& a, const Point& b){
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}

Circle Circumcircle(const Point& a, const Point &b, const Point &c){
    //三角形的外接圆
    Circle ret;
    double a1 = 2 * (a.x - b.x), b1 = 2 * (a.y - b.y), c1 = sqr(a.x) + sqr(a.y) - sqr(b.x) - sqr(b.y);
    double a2 = 2 * (a.x - c.x), b2 = 2 * (a.y - c.y), c2 = sqr(a.x) + sqr(a.y) - sqr(c.x) - sqr(c.y);
    //a1 * x + b1 * y = c1，   a2 * x + b2 * y = c2
    double d = a1 * b2 - a2 * b1, d1 = c1 * b2 - c2 * b1, d2 = a1 * c2 - a2 * c1;
    ret.c.x = d1 / d; ret.c.y = d2 / d; ret.r = Distance(a, ret.c); return ret;
}

Circle Circumcircle(const Point& a, const Point &b){
    Circle ret; ret.c.x = (a.x + b.x) * 0.5; ret.c.y = (a.y + b.y) * 0.5;
    ret.r = Distance(a, ret.c); return ret;
}

Circle Circumcircle(const Point& a){
    Circle ret; ret.c = a; ret.r = 0; return ret;
}

Circle minCircumcircle(Point p[], int n){
    Circle ret;
    if (n == 1){
        ret.c = p[0]; ret.r = 0; return ret;
    }
//  random_shuffle(p, p + n);
```

```
	ret = Circumcircle(p[0],p[1]);
	for (int i = 2; i < n; i++){
		if (Distance(ret.c, p[i]) > ret.r + eps){
			ret = Circumcircle(p[0],p[i]);
			for (int j = 1; j < i; j++){
				if (Distance(ret.c, p[j]) > ret.r + eps){
					ret = Circumcircle(p[j], p[i]);
					for (int k = 0; k < j; k++){
						if (Distance(ret.c, p[k]) > ret.r + eps){
							ret = Circumcircle(p[k], p[j], p[i]);
						}
					}
				}
			}
		}
	}
	return ret;
}

const int maxn = 2000000;
Point p[maxn];

int main(){ //zju1450
	int n;
	while (scanf("%d", &n) != EOF && n){
		for (int i = 0; i < n; i++)
			scanf("%lf%lf", &p[i].x, &p[i].y);
		Circle c = minCircumcircle(p, n);
		printf("%.2f %.2f %.2f\n", c.c.x, c.c.y, c.r);
	}
}
```

## 外接圆最大三角形 – 枚举 1 点，极角排序 - O($n^2$logn)

```
const int maxn = 1000;
const double eps = 1e-8;
const double inf = 1e18;

struct Point { double x, y; } p[maxn], q[maxn];
inline bool equal(const Point& a, const Point &b){ return fabs(a.x - b.x) + fabs(a.y - b.y) < eps;}
```

```
template<Point *p> bool less(Point a, Point b){
	a.x -= p->x; a.y -= p->y;
	b.x -= p->x; b.y -= p->y;
	#define get_quad(x, y) ( (x) >= 0 ?   ( ( (y) >= 0) ? 1 : 2) : ( ( (y) <= 0) ? 3 : 4 ) )
	int aq = get_quad(a.x, a.y), bq = get_quad(b.x, b.y);
	if (aq != bq) return aq < bq; else return a.x * b.y - b.x * a.y > 0;
}

double sqr_circumcircle(Point a, Point b, Point c){
	#define sqr(x) ( (x) * (x) )
	double a1 = 2 * (a.x - b.x), b1 = 2 * (a.y - b.y), c1 = sqr(a.x) + sqr(a.y) - sqr(b.x) - sqr(b.y);
	double a2 = 2 * (a.x - c.x), b2 = 2 * (a.y - c.y), c2 = sqr(a.x) + sqr(a.y) - sqr(c.x) - sqr(c.y);
	double d = a1 * b2 - a2 * b1, d1 = c1 * b2 - c2 * b1, d2 = a1 * c2 - a2 * c1;
	if (fabs(d) < eps) return 0;
	double x = d1 / d, y = d2 / d;
	return sqr(a.x - x) + sqr(a.y - y);
}

int main(){
	int casecnt, n;
	scanf("%d", &casecnt);
	while (casecnt--){
		scanf("%d", &n);
		for (int i = 0; i < n; i++)
			scanf("%lf%lf", &p[i].x, &p[i].y);
		double res2 = 0;
		for (int i = 0; i < n; i++){
			int c = 0;
			q[c++] = p[i];
			for (int j = 0; j < n; j++)
				if (!equal(p[j], p[i]))
					q[c++] = p[j];
			sort(q + 1, q + c, ::less<q>);
			q[c] = q[1];
			for (int j = 1; j < c; j++){
				res2 = max(res2, sqr_circumcircle(q[0], q[j], q[j+1]));
			}
		}
		printf("%.3f\n", sqrt(res2) );
	}
}
```

## 给你一些线段,找到一条直线,要求穿过尽量多的线段 – $O(n^2logn)$

方法：枚举一点，极角排序(2012-4-27) - $O(n^2logn)$
已经通过 2012 金华邀请赛 E
注意用 atan2 要慎重，笔者见过用它精度不够过不了的题

```
#define x first
#define y second
const double eps = 1e-8;

typedef pair<double, double> Point;
Point sub(Point a, Point b){ return make_pair(a.x - b.x, a.y - b.y); }
double cross(Point a, Point b){ return a.x * b.y - b.x * a.y; }
double dot(Point a, Point b){ return a.x * b.x + a.y * b.y; }
int deval(double x){ if ( fabs(x) < eps ) return 0; else return x > 0 ? 1 : -1; }
int dcmp(double x, double y){ return deval(x - y); }

const int maxn = 1000;
Point a[maxn], b[maxn], p[maxn * 2];

struct Elem {
	double arg; //Point p;
	int flag; // 1 for in, -1 for out
} elem[maxn * 2];

Elem make_elem(Point p, int flag){
	Elem ret;
	ret.arg = atan2(p.y, p.x);
	while ( dcmp( ret.arg, 0 ) < 0 ) ret.arg += M_PI; //如果是射线，改为 M_2PI，下同
	while ( dcmp( ret.arg, M_PI ) >= 0 ) ret.arg -= M_PI; //注意不要用  deval( ret.arg ) >= M_PI
	//ret.p = p;
	ret.flag = flag;
	return ret;
}

bool operator < (Elem x, Elem y){
	if ( dcmp(x.arg, y.arg) != 0 ) return x.arg < y.arg;
	// #define quad(p_) ( p_.x >= 0 ? (p_.y >= 0 ? 1 : 4) : (p_.y >= 0 ? 2 : 3) )
	// int qx = quad(x.p), qy = quad(y.p); if (qx != qy) return qx < qy;
	// int crossVal = cross(x.p, y.p); if (crossVal != 0) return crossVal > 0;
	return x.flag > y.flag;
}

void solve(){
        int n; scanf("%d", &n);
        int pcnt = 0, res = 0;
        for (int i = 0; i < n; i++){
                scanf("%lf%lf", &a[i].x, &a[i].y); p[ pcnt++ ] = a[i];
                scanf("%lf%lf", &b[i].x, &b[i].y); p[ pcnt++ ] = b[i];
        }
        sort(p, p + pcnt); pcnt = unique(p, p + pcnt) - p;

        for (int k = 0; k < pcnt; k++){
                Point o = p[k];
                int elemcnt = 0, countAt = 0, current = 0;
                for (int i = 0; i < n; i++){
                        Point oa = sub(a[i], o), ob = sub(b[i], o);
                        int crossVal =    deval( cross( oa, ob ) );
                        int dotVal =    deval( dot(oa, ob) );
                        if ( crossVal == 0 && dotVal <= 0 ) {
                                countAt ++; continue;
                        }

                        if ( crossVal < 0 ) swap(oa, ob);
                        elem[ elemcnt++ ] = make_elem( oa,    1 );
                        elem[ elemcnt++ ] = make_elem( ob, -1 );
                        if ( deval( oa.y ) * deval( ob.y ) < 0 || deval( ob.y ) == 0 && deval( oa.y ) != 0 )
                                ++current; //如果是射线，改为 oay < 0 && oby > 0 || oay < 0 && oby == 0
                }
                sort( elem, elem + elemcnt );
                int mval = current;
                for (int i = 0; i < elemcnt; i++){
                        current += elem[ i ].flag;
                        mval = max( mval, current );
                }
                res = max(res, mval + countAt);
        }
        cout << res << endl;
}
```

## 判断两个正交矩形是否有公共点

```
bool intersect(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4){
        //已经通过 bnu4338
```

```
    //判断两个正交矩形(x1,y1) - (x2,y2), (x3,y3) - (x4,y4)是否有公共点
    if (x1 > x2) swap(x1, x2); if (y1 > y2) swap(y1, y2);
    if (x3 > x4) swap(x3, x4); if (y3 > y4) swap(y3, y4);
    return !(x2 < x3 || y2 < y3 || x4 < x1 || y4 < y1);
}
```