计算几何模版

1.1 基本函数

```
inline int sgn(double n){return fabs(n) < eps ? 0 : (n < 0 ? -1 : 1);}
inline double sqr(double x)
{
    return x * x;
}
inline double Sqrt(double a)
{
    return a <= 0 ? 0 : sqrt(a);
}
int abs(int x)
{
    return x >= 0 ? x : -x;
}
```

2.1 二维点类

```
struct point
{
    double x, y;
    point(){}
    point(double a, double b): x(a), y(b) {}
    friend point operator + (const point &a, const point &b){
        return point(a.x + b.x, a.y + b.y);
    }
    friend point operator - (const point &a, const point &b){
        return point(a.x - b.x, a.y - b.y);
    }
    friend bool operator == (const point &a, const point &b){
        return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0;
    }
    friend point operator * (const point &a, const double &b){
        return point(a.x * b, a.y * b);
    }
    friend point operator * (const double a, const point &b){
        return point(a * b.x, a * b.y);
    }
    friend point operator / (const point &a, const double &b){
        return point(a.x / b, a.y / b);
    }
    double norm(){
        return sqrt(sqr(x) + sqr(y));
    }
};
```

2.2 差积

```
double det(const point &a, const point &b)
{
```

```
    return a.x * b.y - a.y * b.x;
}
```

2.3 点积

```
double dot(const point &a, const point &b)
{
    return a.x * b.x + a.y * b.y;
}
```

2.4 两点之间距离

```
double dist(const point &a, const point &b)
{
    return (a - b).norm();
}
```

2.5 op绕原点逆时针旋转A(弧度)

```
point rotate_point(const point &p, double A)
{
    double tx = p.x, ty = p.y;
    return point(tx * cos(A) - ty * sin(A), tx * sin(A) + ty * cos(A));
}
```

3.1 二维线段直线类

```
struct line
{
    point a, b;
    line(){}
    line(point x, point y): a(x), b(y) {}
};
```

3.2 求p点到线段st的距离(到直线st的距离)

```
double dis_point_segment(const point p, const point s, const point t)
{
    // 似乎这里去掉前两句，就变成了求p点到直线st的距离了?
    // 的确是，前两句是求到端点的距离
    if(sgn(dot(p - s, t - s)) < 0) return (p - s).norm();
    if(sgn(dot(p - t, s - t)) < 0) return (p - t).norm();
    return fabs(det(s - p, t - p) / dist(s, t));
}
```

3.3 求p点到线段st的垂足(大概是直线?)

```
point PointProjLine(const point p, const point s, const point t)
{
    double r = dot((t - s), (p - s)) / dot(t - s, t - s);
    return s + r * (t - s);
```

```
}
```

3.4 判断p点是否在线段st上(包括端点)

```
bool PointOnSegment(point p, point s, point t)
{
    return sgn(det(p - s, t - s)) == 0 && sgn(dot(p - s, p - t)) <= 0;
}
```

3.5 判断a和b是否平行

```
bool parallel(line a, line b)
{
    return !sgn(det(a.a - a.b, b.a - b.b));
}
```

3.6 判断直线a和直线b是否相交

```
// 如果相交则返回true且交点保存在res中
bool line_make_point(line a, line b, point &res)
{
    if(parallel(a, b)) return false;
    double s1 = det(a.a - b.a, b.b - b.a);
    double s2 = det(a.b - b.a, b.b - b.a);
    res = (s1 * a.b - s2 * a.a) / (s1 - s2);
    return true;
}
```

3.7 三角形的有向面积(需*0.5)

```
double cross(const point &o, const point &a, const point &b)
{
    return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}
```

3.8 判断直线a和线段b是否相交

```
// 判断直线a和线段b是否相交 1表示规范相交（只有1个交点） 2表示不规范相交 0表示不相交
(似乎有问题，判断直线和直线的交点时，需要先用上面的line_make_point判断是否有交点，再用下面的line_make_point判断没有交点的情况是不相交还是重合。)
int line_make_point(line a, line b)
{
    int d1, d2;
    d1 = sgn(cross(a.a, a.b, b.a));
    d2 = sgn(cross(a.a, a.b, b.b));
    if((d1 ^ d2) == -2) return 1;
    if(d1 == 0 || d2 == 0) return 2;
    return 0;
}
```

3.9 将直线a沿法向量方向平移距离len得到的直线

```
line move_d(line a, const double &len)
{
    point d = a.b - a.a;
    d = d / d.norm();
    d = rotate_point(d, PI / 2);
    return line(a.a + d * len, a.b + d * len);
}
```

4.1 多边形类

```
int gcd(int x, int y)
{
    return y == 0 ? x : gcd(y, x % y);
}

struct polygon
{
    int n;  // 多边形点数
    point a[N]; // 多边形顶点坐标(按顺时针顺序)  不按顺时针顺序的话会出错，比如面积会变成负数
    polygon(){}
};
```

4.2 计算多边形面积(计算时需注意正负号，即顺逆时针方向)

```
    double area(){  // 计算多边形面积
        double sum = 0;
        a[n] = a[0];
        for(int i = 0; i < n; i ++) sum += det(a[i + 1], a[i]);
        return sum / 2;
    }
```

4.3 判断点是否在多边形内部 O(n)

```
    int Point_In(point t){  // 判断点是否在多边形内部 O(n) 0表示t在多边形外，1表示在内，2表示在边界上
        int num = 0, i, d1, d2, k;
        a[n] = a[0];
        for(i = 0; i < n; i ++){
            if(PointOnSegment(t, a[i], a[i + 1])) return 2;
            k = sgn(det(a[i + 1] - a[i], t - a[i]));
            d1 = sgn(a[i].y - t.y);
            d2 = sgn(a[i + 1].y - t.y);
            if(k > 0 && d1 <= 0 && d2 > 0) num ++;
            if(k < 0 && d2 <= 0 && d1 > 0) num --;
        }
        return num != 0;
    }
```

4.4 返回多边形的重心

```
  point MassCenter(){ // 返回多边形的重心（当多边形面积为0时重心没有定义，需要特别处理）
        point ans = point(0, 0);
        if(sgn(area()) == 0) return ans;
        a[n] = a[0];
        for(int i = 0; i < n; i ++) ans = ans + (a[i] + a[i + 1]) * det(a[i + 1], a[i]);
        return ans / area() / 6.0;
    }
```

4.5 多边形边界上的格点个数

```
    int Border_Int_Point_Num(){ // 多边形边界上的格点个数
        int num = 0;
        a[n] = a[0];
        for(int i = 0; i < n; i ++)
            num += gcd(abs(int(a[i + 1].x - a[i].x)), abs(int(a[i + 1].y - a[i].y)));
        return num;
    }
```

4.6 多边形内的格点个数

```
    int Inside_Int_Point_Num(){ // 多边形内的格点个数
        return int(area()) + 1 - Border_Int_Point_Num() / 2;
    }
```

5.1 凸多边形类

```
struct polygon_convex
{
    vector<point> p;
    polygon_convex(int size = 0){
        p.resize(size);
    }
};
```

5.2 用a中的点求出的凸包（逆时针顺序）

```
bool comp_less(const point &a, const point &b){
    return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0;
}
polygon_convex convex_hull(vector<point> a) // 用a中的点求出的凸包（逆时针顺序）O(nlogn)
{
    polygon_convex res(2 * a.size() + 5);
    sort(a.begin(), a.end(), comp_less);
    a.erase(unique(a.begin(), a.end()), a.end());
```

```
    int m = 0;
    for(int i = 0; i < a.size(); i ++){
        while(m > 1 && sgn(det(res.p[m - 1] - res.p[m - 2], a[i] -
res.p[m - 2])) <= 0) -- m;
        res.p[m ++] = a[i];
    }
    int k = m;
    for(int i = int(a.size()) - 2; i >= 0; i --){
        while(m > k && sgn(det(res.p[m - 1] - res.p[m - 2], a[i] -
res.p[m - 2])) <= 0) -- m;
        res.p[m ++] = a[i];
    }
    res.p.resize(m);
    if(a.size() > 1) res.p.resize(m - 1);
    return res;
}
```

5.3 判断点b是否在凸包a中 O(n) && O(logn) 需要凸包逆时针

```
// 点b是否在凸包a中，true表示点在凸包内部或者在边界上 O(n)
bool containOn(const polygon_convex &a, const point &b)
{
    int n = a.p.size();
#define next(i) ((i + 1) % n)
    int sign = 0;
    for(int i = 0; i < n; i ++){
        int x = sgn(det(a.p[i] - b, a.p[next(i)] - b));
        if(x){
            if(sign){
                if (sign != x) return false;
            }
            else sign = x;
        }
    }
    return true;
}
// 点b是否在凸包a中，true表示点在凸包内部或者在边界上 O(logn) 1表示在凸包内，-1表示
在凸包边界上 0表示在凸包外
int containOlogn(const polygon_convex &a, const point &b)
{
    int n = a.p.size();
    point g = (a.p[0] + a.p[n / 3] + a.p[2 * n / 3]) / 3.0;
    int l = 0, r = n;
    while(l + 1 < r){
        int mid = (l + r) / 2;
        if(sgn(det(a.p[l] - g, a.p[mid] - g)) > 0){
            if(sgn(det(a.p[l] - g, b - g)) >= 0 && sgn(det(a.p[mid] - g,
b - g)) < 0) r = mid;
            else l = mid;
        }
        else{
            if(sgn(det(a.p[l] - g, b - g)) < 0 && sgn(det(a.p[mid] - g,
b - g)) >= 0) l = mid;
            else r = mid;
```

```
        }
    }
    r %= n;
    int z = sgn(det(a.p[r] - b, a.p[l] - b)) - 1;
    if(z == -2) return 1;
    return z;
}
```

6.1 半平面类(计算半平面交的时候需要注意直线的方向)

```
struct halfplane
{
    // ax+by+c<=0
    double a, b, c;
    halfplane(point p, point q){
        a = p.y - q.y;
        b = q.x - p.x;
        c = det(p, q);
    }
    halfplane(double aa, double bb, double cc){
        a = aa; b = bb; c = cc;
    }
};
```

6.2 计算点a带入到直线方程中的函数值

```
double calc(halfplane &L, point &a)
{
    return a.x * L.a + a.y * L.b + L.c;
}
```

6.3 求点a和b连线与半平面L的交点

```
point Intersect(point &a, point &b, halfplane &L)
{
    point res;
    double t1 = calc(L, a), t2 = calc(L, b);
    res.x = (t2 * a.x - t1 * b.x) / (t2 - t1);
    res.y = (t2 * a.y - t1 * b.y) / (t2 - t1);
    return res;
}
```

6.4 将一个凸多边形和一个半平面交 输出一个凸多边形 O(n)

```
polygon_convex cut(polygon_convex &a, halfplane &L)
{
    int n = a.p.size();
    polygon_convex res;
    for(int i = 0; i < n; i ++){
        if(calc(L, a.p[i]) < -eps) res.p.push_back(a.p[i]);
        else{
            int j;
            j = i - 1;
```

```
            if(j < 0) j = n - 1;
            if(calc(L, a.p[j]) < -eps)  // 这里的-eps如果改成eps 在相交的时候，就会留下平行的线或点
                res.p.push_back(Intersect(a.p[j], a.p[i], L));
            j = i + 1;
            if(j == n) j = 0;
            if(calc(L, a.p[j]) < -eps)
                res.p.push_back(Intersect(a.p[i], a.p[j], L));
        }
    }
    return res;
}
```

6.5 给定n个半平面，求出它们的交。 O(nlogn)

```
#include<complex.h>
typedef complex<double> Point;
typedef pair<Point, Point> Halfplane;
const double INF = 10000;
inline double cross(Point a, Point b){return (conj(a) * b).imag();}
inline double dot(Point a, Point b){return (conj(a) * b).real();}
inline double satisfy(Point a, Halfplane p)
{
    return sgn(cross(a - p.first, p.second - p.first)) <= 0;
}
inline bool satisfy2(Point a, Halfplane p)
{
    return sgn(cross(a - p.first, p.second - p.first)) < 0;
}
Point crosspoint(const Halfplane &a, const Halfplane &b)
{
    double k = cross(b.first - b.second, a.first - b.second);
    double k2 = cross(b.first - b.second, a.second - b.second);
    if(sgn(k - k2) == 0) return a.first;
    k = k / (k - k2);
    return a.first + (a.second - a.first) * k;
}
bool Halfcmp(const Halfplane &a, const Halfplane &b)
{
    int res = sgn(arg(a.second - a.first) - arg(b.second - b.first));
    return res == 0 ? satisfy2(a.first, b) : res < 0;    // res == 0 的时候一定要返回false，所以我们这里得重新定义一个比较函数，它和satisfy的区别就是， 不会返回sgn() == 0;
}
// 给定n个半平面，求出它们的交。 O(nlogn) 输入v一组半平面 输出一个凸多边形
vector<Point> halfplaneIntersection(vector<Halfplane> v)
{
    sort(v.begin(), v.end(), Halfcmp);
    deque<Halfplane> q;
    deque<Point> ans;
    q.push_back(v[0]);
    for(int i = 1; i < int(v.size()); i ++){
```

```
        if(sgn(arg(v[i].second - v[i].first) - arg(v[i - 1].second - v[i
- 1].first)) == 0){
            continue;
        }
        while(ans.size() > 0 && !satisfy(ans.back(), v[i])){
            ans.pop_back();
            q.pop_back();
        }
        while(ans.size() > 0 && !satisfy(ans.front(), v[i])){
            ans.pop_front();
            q.pop_front();
        }
        ans.push_back(crosspoint(q.back(), v[i]));
        q.push_back(v[i]);
    }
    while(ans.size() > 0 && !satisfy(ans.back(), q.front())){
        ans.pop_back();
        q.pop_back();
    }
    while(ans.size() > 0 && !satisfy(ans.front(), q.back())){
        ans.pop_front();
        q.pop_front();
    }
    ans.push_back(crosspoint(q.back(), q.front()));
    return vector<Point>(ans.begin(), ans.end());
}
```

6.6 给定两个凸多边形，求它们的交

```
typedef vector<Point> Convex;
// 给定两个凸多边形，求它们的交
Convex convexIntersection(Convex v1, Convex v2)
{
    vector<Halfplane> h;
    for(int i = 0; i < int(v1.size()); i ++)
        h.push_back(Halfplane(v1[i], v1[(i + 1) % v1.size()]));
    for(int i = 0; i < int(v2.size()); i ++)
        h.push_back(Halfplane(v2[i], v2[(i + 1) % v2.size()]));
    return halfplaneIntersection(h);
}
```

6.7 求一个多边形的核

```
// 求一个多边形的核 输入一个多边形，输出一个凸多边形，表示a的核，复杂度取决于半平面交
算法的时间复杂度
const double inf = 1e9;
polygon_convex core(polygon &a)
{
    polygon_convex res;
    res.p.push_back(point(-inf, -inf)); // 设置边界
    res.p.push_back(point(inf, -inf));
    res.p.push_back(point(inf, inf));
    res.p.push_back(point(-inf, inf));
```

```
    int n = a.n;
    for(int i = 0; i < n; i ++){
        halfplane L(a.a[i], a.a[(i + 1) % n]);
        res = cut(res, L);
    }
    return res;
}
```

6.8 凸多边形与直线集交

```
// 给定m条直线和一个n个点的凸包。求这些直线是否和凸包有交点 solve(point P, point
Q) P Q 直线的两个点，输出直线是否与凸包有交点  一条直线与凸包是否有交点复杂度:
O(logn)
inline bool operator < (const point &a, const point &b)
{
    return a.y + eps < b.y || fabs(a.y - b.y) < eps && a.x + eps < b.x;
}

inline double getA(const point &a)  // 凸包顺序下对应的角度也要递增
{
    double res = atan2(a.y, a.x);
    if(res < 0) res += 2 * PI;
    return res;
}
point p[N], hull[N];
int n;
double w[N], sum[N];

// 预处理出n个点的凸包 O(nlogn)
inline void GetHull()
{
    sort(p + 1, p + n + 1);
    int t = 0;
    hull[++ t] = p[1];
    for(int i = 2; i <= n; i ++){
        while(t > 1 && det(hull[t] - hull[t - 1], p[i] - hull[t - 1]) <=
0) -- t;
        hull[++ t] = p[i];
    }
    int bak = t;
    for(int i = n - 1; i >= 1; i --){
        while(t > bak && det(hull[t] - hull[t - 1], p[i] - hull[t - 1])
<= 0) -- t;
        hull[++ t] = p[i];
    }
    n = t - 1;
    for(int i = 1; i <= n; i ++)
        p[i + n] = p[i] = hull[i];
    p[n + n + 1] = p[1];
    for(int i = 1; i <= n; i ++)
        w[i + n] = w[i] = getA(p[i + 1] - p[i]);
    // 预处理有向面积前缀和
    sum[0] = 0;
    for(int i = 1; i <= 2 * n; i ++)
```

```
        sum[i] = sum[i - 1] + det(p[i], p[i + 1]);
}
inline int Find(double x)   // 找第一个角度>x的边
{
    if(x <= w[1] || x >= w[n]) return 1;
    return (upper_bound(w + 1, w + n + 1, x) - (w + 1)) + 1;
}
point P, Q;
inline int getInter(int l, int r)   // 找到第一个和p[l] 不在P->Q向量同侧的点
{
    int sign;
    if(det(Q - P, p[l] - P) < 0) sign = -1;
    else sign = 1;
    while(l + 1 < r){
        int mid = (l + r) / 2;
        if(det(Q - P, p[mid] - P) * sign > 0) l = mid;
        else r = mid;
    }
    return r;
}

inline point Intersect(const point &a, const point &b, const point &c,
const point &d)  // 两直线求交点
{
    double s1 = det(c - a, b - a);
    double s2 = det(d - a, b - a);
    return (c * s2 - d * s1) / (s2 - s1);
}
// 一条直线和凸包的判定
inline bool solve(point P, point Q)
{
    int i = Find(getA(Q - P));
    int j = Find(getA(P - Q));
    if(det(Q - P, p[i] - P) * det(Q - P, p[j] - P) >= 0) return
false;  // 无交点
    else return true;
}
```

7.1 圆类

7.2 求圆与线段（直线）的交点

```
 // 求圆与线段（直线）的交点
double mysqrt(double n)
{
    return sqrt(max(0.0, n));
}
// a,b表示线段（直线）, o圆心， r圆的半径，ret计算出来的交点， &num函数计算出来的交点个数
```

```
void circle_cross_line(point a, point b, point o, double r, point ret[],
int &num)
{
    double x0 = o.x, y0 = o.y;
    double x1 = a.x, y1 = a.y;
    double x2 = b.x, y2 = b.y;
    double dx = x2 - x1, dy = y2 - y1;
    double A = dx * dx + dy * dy;
    double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
    double C = sqr(x1 - x0) + sqr(y1 - y0) - sqr(r);
    double delta = B * B - 4 * A * C;
    num = 0;
    if(sgn(delta) >= 0){
        double t1 = (-B - mysqrt(delta)) / (2 * A);
        double t2 = (-B + mysqrt(delta)) / (2 * A);
        // 把这两个if去掉，就可以计算圆与直线的交点
        ret[num ++] = point(x1 + t1 * dx, y1 + t1 * dy);
        ret[num ++] = point(x1 + t2 * dx, y1 + t2 * dy);
    }
}
```

7.3 求圆与多边形交的面积 O(n)

```
double abs(const point &o)
{
    return sqrt(dot(o, o));
}
point res[N];
double r;
double sector_area(const point &a, const point &b)
{
    double theta = atan2(a.y, a.x) - atan2(b.y, b.x);
    while(theta <= 0) theta += 2 * PI;
    while(theta > 2 * PI) theta -= 2 * PI;
    theta = min(theta, 2 * PI - theta);
    return r * r * theta / 2;
}
double calc(const point &a, const point &b)
{
    point p[2];
    int num = 0;
    int ina = sgn(abs(a) - r) < 0;
    int inb = sgn(abs(b) - r) < 0;
    if(ina){
        if(inb){
            return fabs(det(a, b)) / 2.0;
        }
        else{
            circle_cross_line(a, b, point(0, 0), r, p, num);
            return sector_area(b, p[0]) + fabs(det(a, p[0])) / 2.0;
        }
    }
    else{
        if(inb){
```

```
            circle_cross_line(a, b, point(0, 0), r, p, num);
            return sector_area(p[0], a) + fabs(det(p[0], b)) / 2.0;
        }
        else{
            circle_cross_line(a, b, point(0, 0), r, p, num);
            if(num == 2){
                return sector_area(a, p[0]) + sector_area(p[1], b) +
fabs(det(p[0], p[1])) / 2.0;
            }
            else{
                return sector_area(a, b);
            }
        }
    }
}

// 求圆与简单多边形的交的面积。圆心处于原点。
// 复杂度O(n)  n全局变量，多边形的点数 res全局变量，逆时针存入多边形的所有点，需要保
证res[n]=res[0] r全局变量，圆的半径
double area()
{
    double ret = 0;
    for(int i = 0; i < n; i ++){
        int dcmp = sgn(det(res[i], res[i + 1]));
        if(dcmp != 0){
            ret += dcmp * calc(res[i], res[i + 1]);
        }
    }
    return ret;
}
```

7.4 最小圆覆盖

```
void circle_center(point p0, point p1, point p2, point &cp)
{
    double a1 = p1.x - p0.x, b1 = p1.y - p0.y, c1 = (a1 * a1 + b1 *
b1) / 2;
    double a2 = p2.x - p0.x, b2 = p2.y - p0.y, c2 = (a2 * a2 + b2 *
b2) / 2;
    double d = a1 * b2 - a2 * b1;
    cp.x = p0.x + (c1 * b2 - c2 * b1) / d;
    cp.y = p0.y + (a1 * c2 - a2 * c1) / d;
}
void circle_center(point p0, point p1, point &cp)
{
    cp.x = (p0.x + p1.x) / 2;
    cp.y = (p0.y + p1.y) / 2;
}
point center;
double radius;

bool point_in(const point &p)
{
    return sgn(dist(p, center) - radius) < 0;
```

```
}

// 求一个半径最小的圆覆盖住所有的点。 期望O(n) a需要覆盖的所有的点 n点的个数 输出 center 全局变量，最小覆盖圆的圆心 radius 全局变量，最小覆盖圆的半径
void min_circle_cover(point a[], int n)
{
    radius = 0;
    center = a[0];
    for(int i = 1; i < n; i ++) if(!point_in(a[i])){
        center = a[i]; radius = 0;
        for(int j = 0; j < i; j ++) if(!point_in(a[j])){
            circle_center(a[i], a[j], center);
            radius = dist(a[j], center);
            for(int k = 0; k < j; k ++) if(!point_in(a[k])){
                circle_center(a[i], a[j], a[k], center);
                radius = dist(a[k], center);
            }
        }
    }
}
```

7.5 圆与圆求交

```
point rotate(const point &p, double cost, double sint)
{
    double x = p.x, y = p.y;
    return point(x * cost - y * sint, x * sint + y * cost);
}
// 圆与圆求交， ap,bp 两个圆的圆心， ar,br 两个圆的半径。 输出两个交点（要先确认两圆存在交点）
pair<point, point> crosspoint(point ap, double ar, point bp, double br)
{
    double d = (ap - bp).norm();
    double cost = (ar * ar + d * d - br * br) / (2 * ar * d);
    double sint = sqrt(1. - cost * cost);
    point v = (bp - ap) / (bp - ap).norm() * ar;
    return make_pair(ap + rotate(v, cost, -sint), ap + rotate(v, cost, sint));
}
```

7.6 圆的面积交(精确版)

```
struct circle
{
    long double x, y, r;
}p1, p2;

long double sqr(long double x)
{
    return x * x;
}
long double dis(circle a, circle b)
{
```

```
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}
long double solve(circle a, circle b)
{
    long double d = dis(a, b);
    if(d >= a.r + b.r) return 0;
    if(d <= fabs(a.r - b.r)){
        long double r = min(a.r, b.r);
        return PI * r * r;
    }
    long double ang1 = acos((a.r * a.r + d * d - b.r * b.r) / (2. * a.r
* d)) * 2;
    long double ang2 = acos((b.r * b.r + d * d - a.r * a.r) / (2. * b.r
* d)) * 2;
    double tmp1 = 0.5 * ang1 * a.r * a.r - 0.5 * a.r * a.r * sin(ang1);
    double tmp2 = 0.5 * ang2 * b.r * b.r - 0.5 * b.r * b.r * sin(ang2);
    long double ret = tmp1 + tmp2;
    return ret;
}
```

7.7 圆的面积并

```
// 圆的面积并1
const int NN = N * N + 3 * N;
struct Tcir
{
    double r;
    point o;
};
struct Tinterval
{
    double x, y, Area, mid;
    int type;
    Tcir owner;
    void area(double l, double r){
        double len = sqrt(sqr(l - r) + sqr(x - y));
        double d = sqrt(sqr(owner.r) - sqr(len) / 4.0);
        double angle = atan(len / 2.0 / d);
        Area = fabs(angle * sqr(owner.r) - d * len / 2.0);
    }
}inter[N];
double x[NN];
double l;
int t, tn;
bool compR(const Tcir &a, const Tcir &b)
{
    return a.r > b.r;
}
void Get(Tcir owner, double x, double &l, double &r)
{
    double y = fabs(owner.o.x - x);
    double d = sqrt(fabs(sqr(owner.r) - sqr(y)));
    l = owner.o.y + d;
    r = owner.o.y - d;
```

```
}
void Get_Interval(Tcir owner, double l, double r)
{
    Get(owner, l, inter[tn].x, inter[tn + 1].x);
    Get(owner, r, inter[tn].y, inter[tn + 1].y);
    Get(owner, (l + r) / 2.0, inter[tn].mid, inter[tn + 1].mid);
    inter[tn].owner = inter[tn + 1].owner = owner;
    inter[tn].area(l, r); inter[tn + 1].area(l, r);
    inter[tn].type = 1; inter[tn + 1].type = -1;
    tn += 2;
}
bool comp(const Tinterval &a, const Tinterval &b)
{
    return a.mid > b.mid + eps;
}
void Add(double xx)
{
    x[t ++] = xx;
}
double dist2(const point &a, const point &b)
{
    return sqr(dist(a, b));
}
// 给定n个圆，求它们的面积并。 O(n^3logn) 实际应用中一般远不到这个界。
double getUnion(int n, Tcir a[])
{
    int p = 0;
    sort(a, a + n, compR);
    for(int i = 0; i < n; i ++){
        bool fl = true;
        for(int j = 0; j < i; j ++){
            if(dist2(a[i].o, a[j].o) <= sqr(a[i].r - a[j].r) + 1e-12){
                fl = false;
                break;
            }
        }
        if(fl) a[p ++] = a[i];
    }
    n = p;
    t = 0;
    for(int i = 0; i < n; i ++){
        Add(a[i].o.x - a[i].r);
        Add(a[i].o.x + a[i].r);
        Add(a[i].o.x);
        for(int j = i = 1; j < n; j ++){
            if(dist2(a[i].o, a[j].o) <= sqr(a[i].r + a[j].r) + eps){
                pair<point, point> cross = crosspoint(a[i].o, a[i].r,
a[j].o, a[j].r);
                Add(cross.first.x);
                Add(cross.second.x);
            }
        }
    }
    sort(x, x + t);
    p = 0;
```

```
    for(int i = 0; i < t; i ++){
        if(!i || fabs(x[i] - x[i - 1]) > eps) x[p ++] = x[i];
    }
    t = p;
    double ans = 0;
    for(int i = 0; i + 1 < t; i ++){
        l = x[i], r = x[i + 1];
        tn = 0;
        for(int j = 0; j < n; j ++){
            if(fabs(a[j].o.x - l) < a[j].r + eps && fabs(a[j].o.x - r) <
a[j].r + eps)
                Get_Interval(a[j], l, r);
        }
        if(tn){
            sort(inter, inter + tn, comp);
            int cnt = 0;
            for(int i = 0; i < tn; i ++){
                if(cnt > 0){
                    ans += (fabs(inter[i - 1].x - inter[i].x) +
fabs(inter[i - 1].y - inter[i].y)) * (r - l) / 2.0;
                    ans += inter[i - 1].type * inter[i - 1].Area;
                    ans -= inter[i].type * inter[i].Area;
                }
                cnt += inter[i].type;
            }
        }
    }
    return ans;
}



// 圆的面积并2
struct Circle
{
    point p;
    double r;
    bool operator < (const Circle &o) const{
        if(sgn(r - o.r) != 0) return sgn(r - o.r) == -1;
        if(sgn(p.x - o.p.x) != 0){
            return sgn(p.x - o.p.x) == -1;
        }
        return sgn(p.y - o.p.y) == -1;
    }
    bool operator == (const Circle &o) const{
        return sgn(r - o.r) == 0 && sgn(p.x - o.p.x) == 0 && sgn(p.y -
o.p.y) == 0;
    }
};
inline pair<point, point> crosspoint(const Circle &a, const Circle &b)
{
    return crosspoint(a.p, a.r, b.p, b.r);
}

Circle c[N], tc[N];
int m;
```

```
struct Node
{
    point p;
    double a;
    int d;
    Node(const point &p, double a, int d) : p(p), a(a), d(d) {}
    bool operator < (const Node &o) const{
        return a < o.a;
    }
};

double arg(point p)
{
    return arg(complex<double> (p.x, p.y));
}
// 给定n个圆，求它们的面积并。 O(m^2logm) m全局变量， 圆的个数，tc全局变量，待求面积并的圆
double solve()
{
    sort(tc, tc + m);
    m = unique(tc, tc + m) - tc;
    for(int i = m - 1; i >= 0; i --){
        bool ok = true;
        for(int j = i + 1; j < m; j ++){
            double d = (tc[i].p - tc[j].p).norm();
            if(sgn(d - abs(tc[i].r - tc[j].r)) <= 0){
                ok = false;
                break;
            }
        }
        if(ok) c[n ++] = tc[i];
    }
    double ans = 0;
    for(int i = 0; i < n; i ++){
        vector<Node> event;
        point boundary = c[i].p + point(-c[i].r, 0);
        event.push_back(Node(boundary, -PI, 0));
        event.push_back(Node(boundary, PI, 0));
        for(int j = 0; j < n; j ++){
            if(i == j) continue;
            double d = (c[i].p - c[j].p).norm();
            if(sgn(d - (c[i].r + c[j].r)) < 0){
                pair<point, point> ret = crosspoint(c[i], c[j]);
                double x = arg(ret.first - c[i].p);
                double y = arg(ret.second - c[i].p);
                if(sgn(x - y) > 0){
                    event.push_back(Node(ret.first, x, 1));
                    event.push_back(Node(boundary, PI, -1));
                    event.push_back(Node(boundary, -PI, 1));
                    event.push_back(Node(ret.second, y, -1));
                }
                else{
                    event.push_back(Node(ret.first, x, 1));
                    event.push_back(Node(ret.second, y, -1));
```

```
                }
            }
        }
        sort(event.begin(), event.end());
        int sum = event[0].d;
        for(int j = 1; j < (int)event.size(); j ++){
            if(sum == 0){
                ans += det(event[j - 1].p, event[j].p) / 2;
                double x = event[j - 1].a;
                double y = event[j].a;
                double area = c[i].r * c[i].r * (y - x) / 2;
                point v1 = event[j - 1].p - c[i].p;
                point v2 = event[j].p - c[i].p;
                area -= det(v1, v2) / 2;
                ans += area;
            }
            sum += event[j].d;
        }
    }
    return ans;
}
```

8.1 三维点类

```
class point3
{
public:
    double x, y, z;
    point3() {}
    point3(double x, double y, double z) : x(x), y(y), z(z) {}
    double length() const{  // 向量长度
        return Sqrt(sqr(x) + sqr(y) + sqr(z));
    }
    point3 unit() const;

};
bool operator == (const point3 &a, const point3 &b){
    if(sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0 && sgn(a.z - b.z) ==
0) return 1;
    return 0;
}
point3 operator + (const point3 &a, const point3 &b)
{
    return point3(a.x + b.x, a.y + b.y, a.z + b.z);
}
point3 operator - (const point3 &a, const point3 &b)
{
    return point3(a.x - b.x, a.y - b.y, a.z - b.z);
}
point3 operator * (const point3 &a, double b)
{
    return point3(a.x * b, a.y * b, a.z * b);
}
point3 operator / (const point3 &a, double b)
```

```
{
    return point3(a.x / b, a.y / b, a.z / b);
}
```

8.2 返回单位化的向量

```
point3 point3::unit() const{
    return *this / length();
}
```

8.3 向量a和向量b的叉积，返回的是一个向量

```
point3 det(const point3 &a, const point3 &b)
{
    return point3(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x *
b.y - a.y * b.x);
}
```

8.4 向量a和向量b的点积

```
double dot(const point3 &a, const point3 &b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
```

8.5 向量a,b,c的混合积。返回值除以6就是a,b,c这三个向量所构成的四面体的体积

```
double mix(const point3 &a, const point3 &b, const point3 &c)
{
    return dot(a, det(b, c));
}
```

8.6 三维两点间距离

```
double dist(const point3 &a, const point3 &b)
{
    return Sqrt(sqr(a.x - b.x) + sqr(a.y - b.y) + sqr(a.z - b.z));
}
```

9.1 三位直线、平面类

```
class line3
{
public:
    point3 a, b;
    line3() {}
    line3(point3 a, point3 b) : a(a), b(b) {}
};
struct plane3
{
```

```
public:
    point3 a, b, c;
    plane3() {}
    plane3(point3 a, point3 b, point3 c) : a(a), b(b), c(c) {}
};
```

9.2 线段长度

```
double vlen(point3 p){
    return p.length();
}
```

9.3 零值函数

```
bool zero(double x){
    return fabs(x) < eps;
}
```

9.4 判断三点共线

```
int dots_inlline(point3 p1, point3 p2, point3 p3)
{
    return vlen(det(p1 - p2, p2 - p3)) < eps;
}
```

9.5 判断点在线段内（包含端点）

```
int dot_online_in(point3 p, line3 l)
{
    return zero(vlen(det(p - l.a, p - l.b))) && (l.a.x - p.x) * (l.b.x -
p.x) < eps && (l.a.y - p.y) * (l.b.y - p.y) < eps && (l.a.z - p.z) *
(l.b.z - p.z) < eps;
}
```

9.6 判断点在线段内（不包含端点）

```
int dot_online_ex(point3 p, line3 l)
{
    return dot_online_in(p, l) && (!zero(p.x - l.a.x) || !zero(p.y -
l.a.y) || !zero(p.z - l.a.z)) && (!zero(p.x - l.b.x) || !zero(p.y -
l.b.y) || !zero(p.z - l.b.z));
}
```

9.7 判断平面内两点在直线同侧

```
int same_side(point3 p1, point3 p2, line3 l)
{
    return dot(det(l.a - l.b, p1 - l.b), det(l.a - l.b, p2 - l.b)) >
eps;
}
```

9.8 判断平面内两点在直线异侧

```
int opposite_side(point3 p1, point3 p2, line3 l)
{
    return dot(det(l.a - l.b, p1 - l.b), det(l.a - l.b, p2 - l.b)) < -
eps;
}
```

9.9 判断两直线平行

```
int parallel(line3 u, line3 v)
{
    return vlen(det(u.a - u.b, v.a - v.b)) < eps;
}
```

9.10 判断两直线垂直

```
int perpendicular(line3 u, line3 v)
{
    return zero(dot(u.a - u.b, v.a - v.b));
}
```

9.11 平面法向量

```
point3 pvec(point3 s1, point3 s2, point3 s3){
    return det((s1 - s2), (s2 - s3));
}
```

9.12 判断四点共平面

```
int dots_onplane(point3 a, point3 b, point3 c, point3 d)
{
    return zero(dot(pvec(a, b, c), d - a));
}
```

9.13 判断两条线段是否有交点（包含端点）

```
int intersect_in(line3 u, line3 v)
{
    if(! dots_onplane(u.a, u.b, v.a, v.b)) return 0;
    if(! dots_inlline(u.a, u.b, v.a) || !dots_inlline(u.a, u.b, v.b))
return !same_side(u.a, u.b, v) && !same_side(v.a, v.b, u);
    return dot_online_in(u.a, v) || dot_online_in(u.b, v) ||
dot_online_in(v.a, u) || dot_online_in(v.b, u);
}
```

9.14 判断两条线段是否有交点（不包含端点）

```
int intersect_ex(line3 u, line3 v)
{
```

```
    return dots_onplane(u.a, u.b, v.a, v.b) && opposite_side(u.a, u.b,
v) && opposite_side(v.a, v.b, u);
}
```

9.15 求两直线交点（必须保证共面且不平行）

```
point3 intersection(line3 u, line3 v)
{
    point3 ret = u.a;
    double t = ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) *
(v.a.x - v.b.x)) / ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y)
* (v.a.x - v.b.x));
    ret = ret + (u.b - u.a) * t;
    return ret;
}
```

9.16 点到直线的距离

```
double ptoline(point3 p, line3 l)
{
    return vlen(det(p - l.a, l.b - l.a)) / dist(l.a, l.b);
}
```

9.17 直线到直线的距离，平行时需特别处理

```
double linetoline(line3 u, line3 v)
{
    point3 n = det(u.a - u.b, v.a - v.b);
    return fabs(dot(u.a - v.a, n)) / vlen(n);
}
```

9.18 求两直线夹角的cos值

```
double angle_cos(line3 u, line3 v)
{
    return dot(u.a - u.b, v.a - v.b) / vlen(u.a - u.b) / vlen(v.a -
v.b);
}
```

9.19 判断一个点是否在三角形里（包含边界）

```
int dot_inplane_in(point3 p, plane3 s)
{
    return zero(vlen(det(s.a - s.b, s.a - s.c)) - vlen(det(p - s.a, p -
s.b)) - vlen(det(p - s.b, p - s.c)) - vlen(det(p - s.c, p - s.a)));
}
```

9.20 判断一个点是否在三角形里（不包含边界）

```
int dot_inplane_ex(point3 p, plane3 s)
{
```

```
    return dot_inplane_in(p, s) && vlen(det(p - s.a, p - s.b)) > eps &&
vlen(det(p - s.b, p - s.c)) > eps && vlen(det(p - s.c, p - s.a)) > eps;
}
```

9.21 判断两点在平面同侧(pvec函数在此)

```
point3 pvec(plane3 s)
{
    return det((s.a - s.b), (s.b - s.c));
}
// 判断两点在平面同侧
int same_side(point3 p1, point3 p2, plane3 s)
{
    return dot(pvec(s), p1 - s.a) * dot(pvec(s), p2 - s.a) > eps;
}
```

9.22 判断两点在平面异侧

```
int opposite_side(point3 p1, point3 p2, plane3 s)
{
    return dot(pvec(s), p1 - s.a) * dot(pvec(s), p2 - s.a) < -eps;
}
```

9.23 判断两平面平行

```
int parallel(plane3 u, plane3 v)
{
    return vlen(det(pvec(u), pvec(v))) < eps;
}
```

9.24 判断直线与平面平行

```
int parallel(line3 l, plane3 s)
{
    return zero(dot(l.a - l.b, pvec(s)));
}
```

9.25 判断两平面垂直

```
int perpendicular(plane3 u, plane3 v)
{
    return zero(dot(pvec(u), pvec(v)));
}
```

9.26 判断直线与平面垂直

```
int perpendicular(line3 l, plane3 s)
{
    return vlen(det(l.a - l.b, pvec(s))) < eps;
}
```

9.27 判断线段和三角形是否有交点（包括边界）

```
int intersect_in(line3 l, plane3 s)
{
    return !same_side(l.a, l.b, s) && !same_side(s.a, s.b, plane3(l.a,
l.b, s.c)) && !same_side(s.b, s.c, plane3(l.a, l.b, s.a)) && !
same_side(s.c, s.a, plane3(l.a, l.b, s.b));
}
```

9.28 判断线段和三角形是否有交点（不包括边界）

```
int intersect_ex(line3 l, plane3 s)
{
    return opposite_side(l.a, l.b, s) && opposite_side(s.a, s.b,
plane3(l.a, l.b, s.c)) && opposite_side(s.b, s.c, plane3(l.a, l.b, s.a))
&& opposite_side(s.c, s.a, plane3(l.a, l.b, s.b));
}
```

9.29 求直线与平面的交点

```
point3 intersection(line3 l, plane3 s)
{
    point3 ret = pvec(s);
    double t = (ret.x * (s.a.x - l.a.x) + ret.y * (s.a.y - l.a.y) +
ret.z * (s.a.z - l.a.z)) / (ret.x * (l.b.x - l.a.x) + ret.y * (l.b.y -
l.a.y) + ret.z * (l.b.z - l.a.z));
    ret = l.a + (l.b - l.a) * t;
    return ret;
}
```

9.30 求两平面的交线

```
bool intersection(plane3 pl1, plane3 pl2, line3 &li)
{
    if(parallel(pl1, pl2)) return false;
    li.a = parallel(line3(pl2.a, pl2.b), pl1) ?
intersection(line3(pl2.b, pl2.c), pl1) : intersection(line3(pl2.a,
pl2.b), pl1);
    point3 fa;
    fa = det(pvec(pl1), pvec(pl2));
    li.b = li.a + fa;
    return true;
}
```

9.31 点到平面的距离

```
double ptoplane(point3 p, plane3 s)
{
    return fabs(dot(pvec(s), p - s.a)) / vlen(pvec(s));
}
```

9.32 求两平面的夹角的cos值

```
double angle_cos(plane3 u, plane3 v)
{
    return dot(pvec(u), pvec(v)) / vlen(pvec(u)) / vlen(pvec(v));
}
```

9.33 求平面与直线的夹角的sin值

```
double angle_sin(line3 l, plane3 s)
{
    return dot(l.a - l.b, pvec(s)) / vlen(l.a - l.b) / vlen(pvec(s));
}
```

9.34 将a绕0b向量逆时针旋转弧度angle的结果

```
point3 rotate(point3 a, point3 b, double angle)
{
    point3 e1, e2, e3;
    e3 = b.unit();
    double len = dot(a, e3);
    point3 p = e3 * len;
    e1 = a - p;
    if(e1.length() > (1e-8)) e1.unit();
    e2 = det(e1, e3);
    double x1 = dot(a, e1), y1 = dot(a, e2);
    double x = x1 * cos(angle) - y1 * sin(angle);
    double y = x1 * sin(angle) + y1 * cos(angle);
    return e1 * y + e2 * x + p;
}
```

10.1 其他

10.2 长方形表面两点最短距离

```
int ans;
void turn(int i, int j, int x, int y, int z, int x0, int y0, int L, int
W, int H)
{
    if(z == 0) ans = min(ans, x * x + y * y);
    else{
        if(i >= 0 && i < 2)
            turn(i + 1, j, x0 + L + z, y, x0 + L - x, x0 + L, y0, H, W,
L);
        if(j >= 0 && j < 2)
            turn(i, j + 1, x, y0 + W + z, y0 + W - y, x0, y0 + W, L, H,
W);
        if(i <= 0 && i > -2)
            turn(i - 1, j, x0 - z, y, x - x0, x0 - H, y0, H, W, L);
        if(j <= 0 && j > -2)
            turn(i, j - 1, x, y0 - z, y - y0, x0, y0 - H, L, H, W);
    }
```

```
}
// 给出长方体上两点坐标，求其在长方体表面上的最短路径
// 输入L,W,H 当前长方体的各边长     x1,y1,z1,x2,y2,z2 两点坐标
// 输出  长方体表面上的最短路径的平方     O(1)
int rect_dist(int L, int W, int H, int x1, int y1, int z1, int x2, int
y2, int z2)
{
    if(z1 != 0 && z1 != H){
        if(y1 == 0 || y1 == W)
            swap(y1, z1), swap(y2, z2), swap(W, H);
        else
            swap(x1, z1), swap(x2, z2), swap(L, H);
    }
    if(z1 == H)
        z1 = 0, z2 = H - z2;
    ans = 1 << 30;
    turn(0, 0, x2 - x1, y2 - y1, z2, -x1, -y1, L, W, H);
    return (ans);
}
```

10.3 四面体体积

```
// 给定四面体六条棱的长度，计算此四面体的体积
// 已知四面体棱ab, ac, ad, bc, bd, cd的值分别为l, n, a, m, b, c
double volume(double l, double n, double a, double m, double b, double
c)
{
    double x, y;
    x = 4 * a * a * b * b * c * c - a * a * (b * b + c * c - m * m) * (b
* b + c * c - m * m) - b * b * (c * c + a * a - n * n) * (c * c + a * a
- n * n);
    y = c * c * (a * a + b * b - l * l) * (a * a + b * b - l * l) - (a *
a + b * b - l * l) * (b * b + c * c - m * m) * (c * c + a * a - n * n);
    return sqrt(x - y) / 12;
}
```

10.4 最小球覆盖

```
// 要求一个半径最小的球覆盖住所有的点。
// npoint 全局变量，点数，pt, 全局变量，点的坐标
// RES 全局变量， 球心坐标， radius 全局变量， 球的半径 O(n)
int npoint, nouter;
point3 pt[N], outer[4], RES;
double tmp;
void ball()
{
    point3 q[3];
    double m[3][3], sol[3], L[3], det;
    int i, j;
    RES.x = RES.y = RES.z = radius = 0;
    switch(nouter){
        case 1 : RES = outer[0]; break;
```

```
        case 2 :
            RES.x = (outer[0].x + outer[1].x) / 2;
            RES.y = (outer[0].y + outer[1].y) / 2;
            RES.z = (outer[0].z + outer[1].z) / 2;
            radius = dist(RES, outer[0]);
            break;
        case 3 :
            for(i = 0; i < 2; i ++){
                q[i].x = outer[i + 1].x - outer[0].x;
                q[i].y = outer[i + 1].y - outer[0].y;
                q[i].z = outer[i + 1].z - outer[0].z;
            }
            for(i = 0; i < 2; i ++) for(j = 0; j < 2; j ++)
                m[i][j] = dot(q[i], q[j]) * 2;
            for(i = 0; i < 2; i ++) sol[i] = dot(q[i], q[i]);
            if(fabs(det = m[0][0] * m[1][1] - m[0][1] * m[1][0]) < eps)
return;
            L[0] = (sol[0] * m[1][1] - sol[1] * m[0][1]) / det;
            L[1] = (sol[1] * m[0][0] - sol[0] * m[1][0]) / det;
            RES.x = outer[0].x + q[0].x * L[0] + q[1].x * L[1];
            RES.y = outer[0].y + q[0].y * L[0] + q[1].y * L[1];
            RES.z = outer[0].z + q[0].z * L[0] + q[1].z * L[1];
            radius = dist(RES, outer[0]);
            break;
        case 4 :
            for(i = 0; i < 3; i ++){
                q[i].x = outer[i + 1].x - outer[0].x;
                q[i].y = outer[i + 1].y - outer[0].y;
                q[i].z = outer[i + 1].z - outer[0].z;
                sol[i] = dot(q[i], q[i]);
            }
            for(i = 0; i < 3; i ++) for(j = 0; j < 3; j ++)
                m[i][j] = dot(q[i], q[j]) * 2;
            det = m[0][0] * m[1][1] * m[2][2] + m[0][1] * m[1][2] * m[2]
[0] + m[0][2] * m[2][1] * m[1][0] - m[0][2] * m[1][1] * m[2][0] - m[0]
[1] * m[1][0] * m[2][2] - m[0][0] * m[1][2] * m[2][1];
            if(fabs(det) < eps) return;
            for(j = 0; j < 3; j ++){
                for(i = 0; i < 3; i ++) m[i][j] = sol[i];
                L[j] = (m[0][0] * m[1][1] * m[2][2] + m[0][1] * m[1][2]
* m[2][0] + m[0][2] * m[2][1] * m[1][0] - m[0][2] * m[1][1] * m[2][0] -
m[0][1] * m[1][0] * m[2][2] - m[0][0] * m[1][2] * m[2][1]) / det;
                for(i = 0; i < 3; i ++) m[i][j] = dot(q[i], q[j]) * 2;
            }
            RES = outer[0];
            for(i = 0; i < 3; i ++){
                RES.x += q[i].x * L[i];
                RES.y += q[i].y * L[i];
                RES.z += q[i].z * L[i];
            }
            radius = dist(RES, outer[0]);
    }
}

void minball(int n)
```

```
{
    ball();
    if(nouter < 4){
        for(int i = 0; i < n; i ++){
            if(dist(RES, pt[i]) - radius > eps){
                outer[nouter] = pt[i];
                ++ nouter;
                minball(i);
                -- nouter;
                if(i > 0){
                    point3 Tt = pt[i];
                    memmove(&pt[1], &pt[0], sizeof(point3) * i);
                    pt[0] = Tt;
                }
            }
        }
    }
}

double smallest_ball()
{
    radius = -1;
    for(int i = 0; i < npoint; i ++){
        if(dist(RES, pt[i]) - radius > eps){
            nouter = 1;
            outer[0] = pt[i];
            minball(i);
        }
    }
    return Sqrt(radius);
}
```

10.5 三维凸包

```
// 三维凸包 info 全局变量， 读入的所有点 O(n^2)
#define SIZE(X) (int(X.size()))
int mark[1005][1005];
point3 info[1005];
int cnt;

double area(int a, int b, int c)
{
    return (det(info[b] - info[a], info[c] - info[a])).length();
}

double volume(int a, int b, int c, int d)
{
    return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]);
}

struct Face
{
    int a, b, c;
    Face() {}
```

```
    Face(int a, int b, int c) : a(a), b(b), c(c) {}
    int &operator [] (int k){
        if(k == 0) return a;
        if(k == 1) return b;
        return c;
    }
};

vector<Face> face;
inline void insert(int a, int b, int c)
{
    face.push_back(Face(a, b, c));
}

void add(int v)
{
    vector<Face> tmp;
    int a, b, c;
    cnt ++;
    for(int i = 0; i < SIZE(face); i ++){
        a = face[i][0];
        b = face[i][1];
        c = face[i][2];
        if(sgn(volume(v, a, b, c)) < 0){
            mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[c]
[a] = mark[a][c] = cnt;
        }
        else tmp.push_back(face[i]);
    }
    face = tmp;
    for(int i = 0; i < SIZE(tmp); i ++){
        a = face[i][0];
        b = face[i][1];
        c = face[i][2];
        if(mark[a][b] == cnt) insert(b, a, v);
        if(mark[b][c] == cnt) insert(c, b, v);
        if(mark[c][a] == cnt) insert(a, c, v);
    }
}

int Find()
{
    for(int i = 2; i < n; i ++){
        point3 ndir = det(info[0] - info[i], info[1] - info[i]);
        if(ndir == point3()) continue;
        swap(info[i], info[2]);
        for(int j = i + 1; j < n; j ++){
            if(sgn(volume(0, 1, 2, j)) != 0){
                swap(info[j], info[3]);
                insert(0, 1, 2);
                insert(0, 2, 1);
                return 1;
            }
        }
    }
```

```
    return 0;
}
```

10.6 三维凸包表面积

```
bool operator < (const point3 &a, const point3 &b)
{
    return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) <
0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0 && sgn(a.z - b.z) < 0;
}
// 求出凸包的表面积
double D3_convex()
{
    sort(info, info + n);   //这里的sort内部排序规则应该是上面的operator重定义
    n = unique(info, info + n) - info;
    face.clear();
    random_shuffle(info, info + n);
    if(Find()){
        memset(mark, 0, sizeof(mark));
        cnt = 0;
        for(int i = 3; i < n; i ++) add(i);
        double ans = 0;
        for(int i = 0; i < SIZE(face); i ++){
            point3 p = det(info[face[i][0]] - info[face[i][1]],
info[face[i][2]] - info[face[i][1]]);
            ans += p.length();
        }
        return ans / 2;
    }
    return -1;
}
```

10.7 三角形四心(重心、外心、垂心、内心)

```
point Triangle_Mass_Center(point a, point b, point c)   // 重心
{
    return (a + b + c) / 3;
}

point CircumCenter(point a, point b, point c)   // 外心
{
    point cp;
    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;
    cp.x = a.x + (c1 * b2 - c2 * b1) / d;
    cp.y = a.y + (a1 * c2 - a2 * c1) / d;
    return cp;
}

point Orthocenter(point a, point b, point c)    // 垂心
```

```
{
    return Triangle_Mass_Center(a, b, c) * 3.0 - CircumCenter(a, b, c) *
2.0;
}

point Innercenter(point a, point b, point c)    // 内心(四面体内心据此推导)
{
    point cp;
    double la, lb, lc;
    la = (b - c).norm();
    lb = (c - a).norm();
    lc = (a - b).norm();
    cp.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb + lc);
    cp.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb + lc);
    return cp;
}
```

10.8 四面体内心

```
double S(double p, double a, double b, double c)
{
    return sqrt(p * (p - a) * (p - b) * (p - c));
}
int main()
{
    //fre();
    point3 a[10];
    while(~ scanf("%lf%lf%lf", &a[1].x, &a[1].y, &a[1].z)){
        for(int i = 2; i <= 4; i ++) scanf("%lf%lf%lf", &a[i].x,
&a[i].y, &a[i].z);
        double d[10][10];
        for(int i = 1; i <= 4; i ++){
            for(int j = 1; j <= 4; j ++){
                d[i][j] = (a[i] - a[j]).length();
            }
        }
        double p1 = (d[2][3] + d[3][4] + d[4][2]) / 2, p2 = (d[1][3] +
d[3][4] + d[4][1]) / 2, p3 = (d[1][2] + d[2][4] + d[4][1]) / 2, p4 =
(d[1][2] + d[2][3] + d[3][1]) / 2;
        double s1 = S(p1, d[2][3], d[3][4], d[4][2]);
        double s2 = S(p2, d[1][3], d[3][4], d[4][1]);
        double s3 = S(p3, d[1][2], d[2][4], d[4][1]);
        double s4 = S(p4, d[1][2], d[2][3], d[3][1]);
        double s = s1 + s2 + s3 + s4;
        double v = volume(d[1][2], d[1][3], d[1][4], d[2][3], d[2][4],
d[3][4]);
        if(sgn(s) == 0 || sgn(v) == 0) printf("O O O O\n");
        else{
            double x = (s1 * a[1].x + s2 * a[2].x + s3 * a[3].x + s4 *
a[4].x) / (s);
            double y = (s1 * a[1].y + s2 * a[2].y + s3 * a[3].y + s4 *
a[4].y) / (s);
            double z = (s1 * a[1].z + s2 * a[2].z + s3 * a[3].z + s4 *
a[4].z) / (s);
```

```
            printf("%.4lf %.4lf %.4lf ", x, y, z);
            double r = v * 3 / s;
            printf("%.4lf\n", r);
        }
    }
    return 0;
}
```

10.9 最近点对

```
struct point
{
    LL x, y; int o;
}p[N], tmpp[N];
int n;
LL K(LL x)
{
    return x * x;
}
struct Ans
{
    int x, y;
    LL dis;
    Ans(int x = -1, int y = -1, LL dis = 1e18) : x(x), y(y), dis(dis) {}
    void update(Ans b){
        if(b.dis < dis){
            x = b.x;
            y = b.y;
            dis = b.dis;
        }
    }
};
Ans getDistance(point &a, point &b)
{
    LL dis = K(a.x - b.x) + K(a.y - b.y);
    return Ans(a.o, b.o, dis);
}
bool cmpxy(const point &a, const point &b)
{
    if(a.x != b.x) return a.x < b.x;
    return a.y < b.y;
}
bool cmpy(const point &a, const point &b)
{
    return a.y < b.y;
}
Ans res;
void Closest_Pair(int l, int r)
{
    if(l + 1 == r){
        res.update(getDistance(p[l], p[r]));
    }
    else if(l + 2 == r){
        res.update(getDistance(p[l], p[l + 1]));
        res.update(getDistance(p[l + 1], p[r]));
```

```
        res.update(getDistance(p[l], p[r]));
    }
    else{
        int mid = (l + r) >> 1;
        Closest_Pair(l, mid);
        Closest_Pair(mid + 1, r);
        int g = 0;
        for(int i = l; i <= r; i ++){
            if(K(p[i].x - p[mid].x) < res.dis) tmpp[g ++] = p[i];
        }
        sort(tmpp, tmpp + g, cmpy);
        for(int i = 0; i < g; i ++){
            for(int j = i + 1; j < g && K(tmpp[j].y - tmpp[i].y) <
res.dis; j ++){
                res.update(getDistance(tmpp[j], tmpp[i]));
            }
        }
    }
}

int main()
{
    while(~ scanf("%d", &n)){
        for(int i = 0; i < n; i ++){
            scanf("%lld", &p[i].x);
        }
        p[0].y = p[0].x; p[0].x = 0;
        for(int i = 1; i < n; i ++){
            p[i].y = p[i].x + p[i - 1].y;
            p[i].x = i;
        }
        sort(p, p + n, cmpxy);
        for(int i = 0; i < n; i ++) p[i].o = i;
        res = Ans(); Closest_Pair(0, n - 1);
        LL ans = res.dis;  // 此处的ans是最近距离的平方
        printf("%lld\n", ans);
    }
    return 0;
}
```

10.10 平面最小曼哈顿距离生成树 O(nlogn)

```
inline int lowbit(const int &x) {return x & -x;}
struct Edge
{
    int u, v, c;
    Edge(int _u = 0, int _v = 0, int _c = 0) : u(_u), v(_v), c(_c) {}
}edge[N * 4];

inline bool operator < (const Edge &a, const Edge &b) {return a.c <
b.c;}
struct node
{
    int key, id;
```

```
    node(int _k = 0, int _i = 0) : key(_k), id(_i) {}
}Tree[N];
inline bool operator < (const node &a, const node &b)
{
    return a.key < b.key;
}

int idx[N], idy[N], bak[N];
int xx[N], yy[N], id[N], father[N];
int find(const int &x)
{
    return father[x] == x ? x : father[x] = find(father[x]);
}
inline bool cmp1(const int &i, const int &j)
{
    return xx[i] - yy[i] > xx[j] - yy[j] || xx[i] - yy[i] == xx[j] -
yy[j] && yy[i] > yy[j];
}
inline bool cmp2(const int &i, const int &j)
{
    return xx[i] - yy[i] < xx[j] - yy[j] || xx[i] - yy[i] == xx[j] -
yy[j] && yy[i] > yy[j];
}
inline bool cmp3(const int &i, const int &j)
{
    return xx[i] + yy[i] < xx[j] + yy[j] || xx[i] + yy[i] == xx[j] +
yy[j] && yy[i] > yy[j];
}
inline bool cmp4(const int &i, const int &j)
{
    return xx[i] + yy[i] < xx[j] + yy[j] || xx[i] + yy[i] == xx[j] +
yy[j] && yy[i] < yy[j];
}
inline void Process(int x[], int idx[], int n)
{
    for(int i = 0; i < n; i ++) bak[i] = x[i];
    sort(bak, bak + n, greater<int>());
    int p = unique(bak, bak + n) - bak;
    for(int i = 0; i < n; i ++)
        idx[i] = lower_bound(bak, bak + p, x[i], greater<int>()) - bak +
1;
}
inline void add_edge(int &N, const int &u, const int &v)
{
    edge[N ++] = Edge(u, v, abs(xx[u] - xx[v]) + abs(yy[u] - yy[v]));
}

inline int get_min(const int &p)
{
    node tmp(INF);
    for(int i = p; i; i ^= lowbit(i))
        if(Tree[i].id != -1)
            tmp = min(tmp, Tree[i]);
    return tmp.key == INF ? -1 : tmp.id;
}
```

```
inline void insert(const int &n, const int &p, const node &it)
{
    for(int i = p; i <= n; i += lowbit(i))
        if(Tree[i].id == -1 || it < Tree[i]) Tree[i] = it;
}
inline long long Minimum(int x[], int y[], int n)   // 返回最小生成树的权值
和
{
    Process(x, idx, n);
    Process(y, idy, n);
    int N = 0;
    for(int i = 0; i < n; i ++) id[i] = i;
    sort(id, id + n, cmp1);
    for(int i = 1; i <= n; i ++) Tree[i].id = -1;
    for(int i = 0; i < n; i ++){
        int u = id[i], v = get_min(idy[u]);
        if(v != -1)
            add_edge(N, u, v);
        insert(n, idy[u], node(x[u] + y[u], u));
    }
    for(int i = 0; i < n; i ++) id[i] = i;
    sort(id, id + n, cmp2);
    for(int i = 1; i <= n; i ++) Tree[i].id = -1;
    for(int i = 0; i < n; i ++){
        int u = id[i], v = get_min(idx[u]);
        if(v != -1) add_edge(N, u, v);
        insert(n, idx[u], node(x[u] + y[u], u));
    }
    for(int i = 0; i < n; i ++) id[i] = i;
    sort(id, id + n, cmp3);
    for(int i = 1; i <= n; i ++) Tree[i].id = -1;
    for(int i = 0; i < n; i ++){
        int u = id[i], v = get_min(idy[u]);
        if(v != -1) add_edge(N, u, v);
        insert(n, idy[u], node(-x[u] + y[u], u));
    }
    for(int i = 0; i < n; i ++) id[i] = i;
    sort(id, id + n, cmp4);
    for(int i = 1; i <= n; i ++) Tree[i].id = -1;
    for(int i = 0; i < n; i ++){
        int u = id[i], v = get_min(idx[u]);
        if(v != -1) add_edge(N, u, v);
        insert(n, idx[u], node(x[u] - y[u], u));
    }
    sort(edge, edge + N);
    for(int i = 0; i < n; i ++) father[i] = i;
    long long res = 0;
    for(int i = 0; i < N; i ++){
        int u = find(edge[i].u), v = find(edge[i].v);
        if(u != v){
            father[u] = v;
            res += edge[i].c;
        }
    }
    return res;
```

```
}
```

10.11 最大空凸包

```
// 给定n个点，求出最大空凸包。 O(n^3) dott全局变量，所有点的坐标，n全局变量，点数
struct Vector
{
    double x, y;
};
inline Vector operator - (Vector a, Vector b)
{
    Vector c;
    c.x = a.x - b.x;
    c.y = a.y - b.y;
    return c;
}

inline bool operator < (Vector a, Vector b)
{
    return sgn(b.y - a.y) > 0 || sgn(b.y - a.y) == 0 && sgn(b.x - a.x) >
0;
}

inline double Length(Vector a)
{
    return sqrt(sqr(a.x) + sqr(a.y));
}

inline double cross(Vector a, Vector b)
{
    return a.x * b.y - a.y * b.x;
}
Vector dott[N], List[N];
double opt[N][N];
int seq[N];
int len;
double ANS;
// int n;
bool Compare(Vector a, Vector b)
{
    int tmp = sgn(cross(a, b));
    if(tmp) return tmp > 0;
    tmp = sgn(Length(b) - Length(a));
    return tmp > 0;
}
void solve(int vv)
{
    int t, i, j, _len;
    for(i = len = 0; i < n; i ++)
        if(dott[vv] < dott[i]) List[len ++] = dott[i] - dott[vv];
    for(i = 0; i < len; i ++)
        for(j = 0; j < len; j ++)
```

```
            opt[i][j] = 0;
    sort(List, List + len, Compare);
    double v;
    for(t = 1; t < len; t ++){
        _len = 0;
        for(i = t - 1; i >= 0 && sgn(cross(List[t], List[i])) == 0; i
--);
        while(i >= 0){
            v = cross(List[i], List[t]) / 2;
            seq[_len ++] = i;
            for(j = i - 1; j >= 0 && sgn(cross(List[i] - List[t],
List[j] - List[t])) > 0; j --);
            if(j >= 0) v += opt[i][j];
            ANS = max(ANS, v);
            opt[t][i] = v;
            i = j;
        }
        for(i = _len - 2; i >= 0; i --)
            opt[t][seq[i]] = max(opt[t][seq[i]], opt[t][seq[i + 1]]);
    }
}
// 返回最大空凸包的大小
double Empty()
{
    ANS = 0;
    for(int i = 0; i < n; i ++) solve(i);
    return ANS;
}
```

10.12 给定n个圆(1e5)问是否有交集(随机生成的点才能过)

```
inline int sgn(double x)
{
    return fabs(x) < eps ? 0 : (x < 0 ? -1 : 1);
}

struct circle
{
    double x, y, r;
}a[N];
double l, r, mid, ll, rr;
double bot, top;
int n;

int cal(double x)
{
    bool lft = 0, rgt = 0;
    for(int i = 1; i <= n; i ++){
        if(sgn(a[i].x + a[i].r - x) > 0 && sgn(a[i].x - a[i].r - x) < 0)
{
            continue;
        }
        if(sgn(a[i].x - x) < 0) lft = 1;
        else rgt = 1;
```

```
    }
    if(lft == 0 && rgt == 0) return 0;
    else if(lft == 1 && rgt == 1) return -1;
    else if(lft == 1 && rgt == 0) return 1;
    else return 2;
}

int cal2(double y)
{
    bool lft = 0, rgt = 0;
    for(int i = 1; i <= n; i ++){
        if(sgn(a[i].y + a[i].r - y) > 0 && sgn(a[i].y - a[i].r - y) < 0)
{
            continue;
        }
        if(sgn(a[i].y - y) < 0) lft = 1;
        else rgt = 1;
    }
    if(lft == 0 && rgt == 0) return 0;
    else if(lft == 1 && rgt == 1) return -1;
    else if(lft == 1 && rgt == 0) return 1;
    else return 2;
}

#include<stdlib.h>
#include<time.h>

double sqr(double x)
{
    return x * x;
}
double dist(circle a, circle b)
{
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}
bool judge(circle p)
{
    for(int i = 1; i <= n; i ++){
        if(dist(p, a[i]) > a[i].r) return 0;
    }
    return 1;
}


int main()
{
    while(~ scanf("%d", &n)){
        for(int i = 1; i <= n; i ++){
            scanf("%lf%lf%lf", &a[i].x, &a[i].y, &a[i].r);
        }
        l = -INF; r = INF;
        int flag = 1;
        ll = -INF; rr = INF;
        for(int tim = 1; tim <= 100; tim ++){
            mid = (ll + rr) / 2;
```

```
            int tmp = cal(mid);
            if(tmp == 0) rr = mid;  // 全部相交
            else if(tmp == -1){flag = 0; break;}    // 左右都有不相交的圆
            else if(tmp == 1) rr = mid; // 只有左边有不相交的圆
            else if(tmp == 2) ll = mid; // 只有右边有不相交的圆
        }
        if(!flag){puts("NO"); continue;}
        l = ll;
        ll = -INF; rr = INF;
        for(int tim = 1; tim <= 100; tim ++){
            mid = (ll + rr) / 2;
            int tmp = cal(mid);
            if(tmp == 0) ll = mid;
            else if(tmp == -1){flag = 0; break;}
            else if(tmp == 1) rr = mid;
            else if(tmp == 2) ll = mid;
        }
        if(!flag){puts("NO"); continue;}
        r = rr;

        bot = -INF; top = INF;
        ll = -INF; rr = INF;
        for(int tim = 1; tim <= 100; tim ++){
            mid = (ll + rr) / 2;
            int tmp = cal2(mid);
            if(tmp == 0) rr = mid;  // 全部相交
            else if(tmp == -1){flag = 0; break;}    // 左右都有不相交的圆
            else if(tmp == 1) rr = mid; // 只有左边有不相交的圆
            else if(tmp == 2) ll = mid; // 只有右边有不相交的圆
        }
        if(!flag){puts("NO"); continue;}
        bot = ll;
        ll = -INF; rr = INF;
        for(int tim = 1; tim <= 100; tim ++){
            mid = (ll + rr) / 2;
            int tmp = cal2(mid);
            if(tmp == 0) ll = mid;
            else if(tmp == -1){flag = 0; break;}
            else if(tmp == 1) rr = mid;
            else if(tmp == 2) ll = mid;
        }
        if(!flag){puts("NO"); continue;}
        top = rr;
        flag = 0;
        for(int tim = 1; tim <= 100; tim ++){
            circle p;
            p.x = fmod(double(rand()), (r - l)) + l;
            p.y = fmod(double(rand()), (top - bot)) + bot;
            if(judge(p)) {flag = 1; break;}
        }
        if(flag) puts("YES");
        else puts("NO");
    }
    return 0;
```

```
}
```

10.13 单位圆最多覆盖多少个点(复杂度小于n^3)

```
int n, ctp;
double r;
inline int sgn(double n){return fabs(n) < eps ? 0 : (n < 0 ? -1 : 1);}
struct point
{
    double x, y;
}a[N];
int dcmp(double x)
{
    return (x > eps) - (x < -eps);
}
double sqr(double x)
{
    return x * x;
}
double dist(point a, point b)
{
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}
struct A
{
    double site;
    int num;
}b[N << 2];
int cmp2(A a, A b)
{
    if(!dcmp(a.site - b.site)) return -(a.num - b.num);
    return dcmp(a.site - b.site);
}
int cmp(const void *a, const void *b)//角度区间排序
{
    if(!dcmp((*(A*)a).site - (*(A*)b).site))
        return -((*(A*)a).num - (*(A*)b).num);
    return dcmp((*(A*)a).site - (*(A*)b).site);
}
void cal(double &x)
{
    while(x + PI < eps) x += 2 * PI;
    while(x - PI > eps) x -= 2 * PI;
}
void add(double st, double ed)
{
    cal(st), cal(ed);
    if(sgn(st - ed - eps) > 0) add(st, PI), add(-PI + eps * 2, ed);
    else{
        b[ctp].site = st, b[ctp].num = 1; ++ ctp;
        b[ctp].site = ed, b[ctp].num = -1; ++ ctp;
    }
```

```
}
int solve()
{
    int cnt = 0, ans = 0;
    double dis, ang, ac, d = 2 * r; // 这里的d变成 2 * (r + 0.0001) g++
wa, c++ re... wkc说计算几何就是玄学- -
    for(int i = 0; i < n; i ++){
        for(int j = ctp = 0; j < n; j ++){
            if(j != i && sgn((dis = dist(a[i], a[j])) - d) < 0){
                ang = atan2(a[j].y - a[i].y, a[j].x - a[i].x);
                ac = acos(dis * 0.5 / r);
                add(ang - ac, ang + ac);
            }
        }
        //sort(b, b + ctp, cmp2);   不知道怎么更改
        qsort(b, ctp, sizeof(A), cmp);
        for(int j = cnt = 0; j < ctp; j ++){
            gmax(ans, cnt += b[j].num);
        }
    }
    return ans + 1;
}
int main()
{
    r = 1.0;
    while(scanf("%d", &n), n){
        for(int i = 0; i < n; i ++) scanf("%lf%lf", &a[i].x, &a[i].y);
        printf("%d\n", solve());
    }
    return 0;
}
```

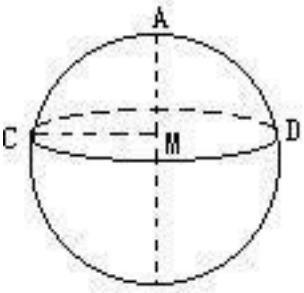
10.14 球缺

一个球被平面截下的一部分叫做球缺。截面叫做球缺的底面，垂直于截面的直径被截下的线段长叫做球缺的高。

球冠的面积=2πRH（不包括截面的面积）

球缺体积公式=(π/3)(3R-H)*H^2（R是球的半径,H是球缺的高）

球缺质心：匀质球缺的质心位于它的中轴线上，并且与底面的距离为：



c = (4R-H)H/(12R-4H) = (d^2+2H^2)H/(3d^2+4H^2)

（其中，H为球缺的高，R为大圆半径，d为球缺的底面直径。）

10.15 平面两点集中最近点对距离

```
const int N = 200000+1;
const double INF = 1e100;
struct Point{
    double x, y;
    bool flag;
};
Point m[N + 1];
int tmp[N + 1];
double dis(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y -
p2.y));
}
double minL(double l1, double l2)
{
    return l1 <= l2 ? l1 : l2;
}
bool cmpy(int a, int b)
{
    return m[a].y < m[b].y;
}
bool cmpx(Point a, Point b)
{
    return a.x < b.x;
}
double getMinLen(Point *s, int left, int right)
{
    double rs = INF;
    if(left == right)
        return rs;
    if(left + 1 == right){
        if(s[left].flag == s[right].flag) return rs;
        return dis(s[left], s[right]);
    }
    int mid = (left + right) >> 1;
    rs = getMinLen(s, left, mid);
    rs = minL(rs, getMinLen(s, mid + 1, right));//rs现在是两边中最点值了
    int i, j, num = 0;
    for(i = left; i <= right; ++ i){
        if(fabs(s[i].x - s[mid].x) <= rs)//过滤掉不可能是最近点对的点
            tmp[num ++] = i;
    }
    sort(tmp, tmp + num, cmpy);//按y坐标排序，减少比较次数
    double d = INF;
    for(i = 0; i < num; ++ i){
        for(j = i + 1; j < num; ++ j){
            if(fabs(s[tmp[i]].y - s[tmp[j]].y) >= rs)//i，j两点的y坐标已经
大于rs,不需要在遍历了
                break;
            if(s[tmp[i]].flag != s[tmp[j]].flag && (d = dis(s[tmp[i]],
s[tmp[j]])) < rs)
                rs = d;
```

```
        }
    }
    //  cout<<left<<" "<<right<<" "<<rs<<endl;
    return rs;
}
int main()
{
    int t, n, i;
    scanf("%d", &t);
    while(t--){
        scanf("%d", &n);
        for(i = 0; i < n; ++ i){
            scanf("%lf%lf", &m[i].x, &m[i].y);
            m[i].flag = 0;
        }
        for(i = 0; i < n; ++ i){
            scanf("%lf%lf",&m[i + n].x, &m[i + n].y);
            m[i + n].flag = 1;
        }
        n <<= 1;
        sort(m, m + n, cmpx);
        double ans = getMinLen(m, 0, n - 1);
        cout << setiosflags(ios::fixed) << setprecision(3) << ans <<
endl;
    }
    return 0;
}
```

10.16 矩形面积并+矩形周长并(wkc模版)

```
int n;
int val[N], topval;
#define rt 1, 1, topval - 1
struct B
{
      int y, l, r, v;
      const bool operator < (const B&b)const
      {
            if (y != b.y)return y < b.y;
            return v > b.v; // add first sub second
      }
}b[N * 2];
int flag[1 << 16];
int sum[1 << 16];
int num[1 << 16];
bool lft[1 << 16];
bool rgt[1 << 16];
int len[1 << 16];
void build(int o, int l, int r)
{
      flag[o] = sum[o] = num[o] = lft[o] = rgt[o] = 0;
      len[o] = val[r + 1] - val[l];
      if (l == r)return;
      int mid = (l + r) >> 1;
      build(lson);
```

```
	build(rson);
}
void pushup(int o)
{
	if (flag[o])
	{
		sum[o] = len[o];
		num[o] = 2;
		lft[o] = rgt[o] = 1;
	}
	else if (len[o] != 1)
	{
		sum[o] = sum[ls] + sum[rs];
		num[o] = num[ls] + num[rs] - (rgt[ls] && lft[rs]) * 2;
		lft[o] = lft[ls];
		rgt[o] = rgt[rs];
	}
	else sum[o] = num[o] = lft[o] = rgt[o] = 0;
}
int L, R, V;
void modify(int o, int l, int r)
{
	if (l >= L&&r <= R)flag[o] += V;
	else
	{
		int mid = (l + r) >> 1;
		if (L <= mid)modify(lson);
		if (R > mid)modify(rson);
	}
	pushup(o);
}
int main()
{
	while (~scanf("%d", &n))
	{
		topval = 0;
		int g = 0;
		for (int i = 1; i <= n; ++i)
		{
			int y1, x1, y2, x2;
			scanf("%d%d%d%d", &y1, &x1, &y2, &x2);
			b[++g] = { y1,x1,x2,1 };
			b[++g] = { y2,x1,x2,-1 };
			val[++topval] = x1;
			val[++topval] = x2;
		}
		sort(b + 1, b + g + 1);
		sort(val + 1, val + topval + 1);
		topval = unique(val + 1, val + topval + 1) - val - 1;
		build(rt);
		LL area = 0;
		LL peri = 0;
		for (int i = 1; i <= g; )
		{
			area += sum[1] * (b[i].y - b[i - 1].y);
```

```
                peri += num[1] * (b[i].y - b[i - 1].y);
                int Y = b[i].y;
                //把同一纵坐标的所有加都处理
                for (; i <= g && b[i].y == Y && b[i].v == 1; ++i)
                {
                     b[i].l = lower_bound(val + 1, val + topval + 1,
b[i].l) - val;
                     b[i].r = lower_bound(val + 1, val + topval + 1,
b[i].r) - val - 1;
                     int len1 = sum[1];
                     L = b[i].l; R = b[i].r; V = b[i].v;
                     modify(rt);
                     int len2 = sum[1];
                     if (len2 > len1)peri += len2 - len1;
                }
                //把同一纵坐标的所有减都处理
                for (; i <= g && b[i].y == Y && b[i].v == -1; ++i)
                {
                     b[i].l = lower_bound(val + 1, val + topval + 1,
b[i].l) - val;
                     b[i].r = lower_bound(val + 1, val + topval + 1,
b[i].r) - val - 1;
                     int len1 = sum[1];
                     L = b[i].l; R = b[i].r; V = b[i].v;
                     modify(rt);
                     int len2 = sum[1];
                     if (len2 < len1)peri += len1 - len2;
                }
           }
           //printf("%lld\n", area);
           printf("%lld\n", peri);
      }
      return 0;
}
```

10.17 求平面上锐角三角形的个数(n^2logn)

数一数锐角的数量A和直角+钝角的数量B，那么答案就是(A-2B)/3。

```
const double PI2 = acos(-1.0) * 2;
const double eps = 1e-8;
int n;
struct P
{
    int x, y;
}p[N];
inline int sgn(double n){return fabs(n) < eps ? 0 : (n < 0 ? -1 : 1);}

struct L
{
    int x, y;
    double atan2;
    bool operator < (const L &b)const
    {
        return atan2 < b.atan2;
```

```
    }
}a[N * 3];
bool ok(int i, int j)//我们认为j在i的逆时针方向
{
    return sgn(a[j].atan2 - a[i].atan2 - PI2/4) < 0;
}
int main()
{
    int casei = 0;
    while (scanf("%d", &n), n)
    {
        for (int i = 1; i <= n; ++i)scanf("%d%d", &p[i].x, &p[i].y);
        int ans = n * (n - 1) * (n - 2ll) / 6;
        for (int i = 1; i <= n; ++i)
        {
            int m = 0;
            for (int j = 1; j <= n; ++j)if (j != i)
            {
                a[++m].x = p[j].x - p[i].x;
                a[m].y = p[j].y - p[i].y;
                a[m].atan2 = atan2(a[m].y, a[m].x);
            }
            sort(a + 1, a + m + 1);
            for (int i = m + 1; i <= 2 * m; ++i)a[i] = a[i - m],
a[i].atan2 += PI2;
            for (int i = 2 * m + 1; i <= 3 * m; ++i)a[i] = a[i - m],
a[i].atan2 += PI2;
            //固定第一条边，找出改变前后的成锐角的极限边
            int k = m + 1;
            int l = m + 1, r = m + 1;
            while (k - l < m - 1 && ok(l - 1, k))--l;
            while (r - l < m - 1 && ok(k, r + 1))++r;
            int tmp = 0;
            tmp += m - (r - l + 1);
            for (++k; k <= m + m; ++k)
            {
                while (k - l >= m || !ok(l, k))++l;
                while (r - l < m - 1 && ok(k, r + 1))++r;
                tmp += m - (r - l + 1);
            }
            ans -= tmp / 2;
        }
        printf("Scenario %d:\n", ++ casei);
        printf("There are %d sites for making valid tracks\n", ans);
    }
    return 0;
}
```

10.18 最近平面圆对(二分加扫描线)

10.19 爬山算法（求多边形费马点(包括三角形费马点求法)）

费马点定义如下： 寻找多边形内一个点，使得多边形上所有顶点到该点的距离之和最小。
三角形的费马点求法如下：
若有一个角大于等于120度，那么这个点就是费马点。
若不存在，那么对三角形ABC，任取两条边（这里取AB、BC两条），向三角形外做等边三角形，得到的新的顶点成为C'和A'，那么AA'和CC'的交点即是费马点。

```
const double delta = 0.98;
const int init_T = 100;
const double INF = 1e30;

struct point
{
    double x, y;
}a[N];
int n;

const int dx[4] = {1, 1, -1, -1}, dy[4] = {1, -1, 1, -1};

double sqr(double x)
{
    return x * x;
}

double dist(point a, point b)
{
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}

double sum(point p)
{
    double ans = 0;
    for(int i = 1; i <= n; i ++){
        ans += dist(a[i], p);
    }
    return ans;
}

void solve()
{
    point p = a[1];
    double t = init_T;
    double ans = INF;
    while(t > eps){
        bool flag = 1;
        while(flag){
            flag = 0;
            point pp;
            for(int i = 0; i < 4; i ++){
```

```
                pp.x = p.x + dx[i] * t;
                pp.y = p.y + dy[i] * t;
                double cur = sum(pp);
                if(cur < ans){
                    ans = cur;
                    flag = 1;
                    p = pp;
                }
            }
        }
        t *= delta;
    }
    printf("%.0f\n", ans);
}
int main()
{
    while(~ scanf("%d", &n)){
        for(int i = 1; i <= n; i ++){
            scanf("%lf%lf", &a[i].x, &a[i].y);
        }
        solve();
    }
    return 0;
}
```

10.20 convex hull trick

```
struct point
{
    double x;
    int o;
}inter[N];
int n;
LL a[N], b[N];
LL f[N];

double cal(int i, int j)
{
    // 直线 y1 = f[i] + b[i] * x1, y2 = f[j] + b[j] * x2;
    // 因为这里保证了b[i] != b[j] 所以直接求出交点即可
    double x = 1.0 * (f[j] - f[i]) / (b[i] - b[j]);
    return x;
}

int main()
{
    while(~ scanf("%d", &n)){
        for(int i = 1; i <= n; i ++) scanf("%lld", &a[i]);
        for(int i = 1; i <= n; i ++) scanf("%lld", &b[i]);
        f[1] = 0;
        inter[1].x = 0; // 我们用inter保存交点坐标及相应的直线编号，当当前直线与前一条直线计算出的交点坐标小于前一个交点坐标（即inter里存储的最后一个交点坐标）时，前一个交点坐标就出栈
        inter[1].o = 1;
```

```
        int cnt = 1;
        for(int i = 2; i <= n; i ++){
            // f[i] = max(f[j] + b[j] * a[i]);

            int l = 1, r = cnt;
            while(l < r){   // 我们用二分找到最后一个坐标小于等于a[i]的交点，a[i]所取的最值一定是这个交点所对应的直线上的
                int mid = (l + r + 1) >> 1;
                if(inter[mid].x > a[i])r = mid - 1;
                else l = mid;
            }
            int o = inter[l].o;
            f[i] = f[o] + b[o] * a[i];
            while(1){
                double pos = cal(i, inter[cnt].o);  // 计算交点坐标
                if(pos <= inter[cnt].x) cnt --;
                else{
                    inter[++ cnt].o = i;
                    inter[cnt].x = pos;
                    break;
                }
            }
        }
        printf("%lld\n", f[n]);
    }
    return 0;
}

/*
```

题意:

森林里有n(n <= 1e5)棵树，每棵树有一个高度a[i]和一个价值b[i];

刚开始我们有一把斧子，我们可以用这把斧子砍任意一棵树，每次可以使得一棵树的高度下降1.

但是每砍过一次之后，我们都需要花一些钱修复这把斧子，否则就不能继续砍树。

我们每次修复斧子需要花费的金钱是当前砍完（即高度为0）的树中id最大的一棵树的价值。

现在问我们至少要花费多少金钱才能砍完所有的树。

数据保证a[1] = 1, b[n] = 0, a[1] < a[2] < ... < a[n], b[1] > b[2] > ... b[n]

0 <= a[i] <= 1e9

0 <= b[i] <= 1e9

类型:

计算几何_conevx hull trick(凸包把戏?233)_斜率dp?

分析:

首先我们可以很容易想出一个O(n^2)的dp转移方程。

令f[i]表示砍掉第i棵树最少需要花费的金钱。(为什么不说是砍完呢，因为我们可以保留一些树先不砍，以最少金钱砍完第n棵树再去砍前面剩下来的树一定是最优的决策)

f[i] = min(f[j] + b[j] * a[i]) (1 <= j < i)

这时候我们就可以用一个叫做"convex hull trick"的东西来对这个dp进行优化。

详情参见：http://wcipeg.com/wiki/Convex_hull_trick

噗，似乎这就是斜率优化dp?
好开心呀，以前还试图学习过斜率优化dp，但是当时没有学会，现在终于明白啦 ！

*/

10.21 凸螺旋线构造方法

凸螺旋线可以通过如下方法构造：

1. 从一个特定的端点开始（比如给定方向上的最小点）， 这里取有最小 x 坐标的点。
2. 通过那个点构造一条铅垂线。
3. 按照一个给定的方向旋转线（总保持顺时针或者是逆时针方向）， 直到线“击” 出另一个顶点。
4. 将两个点用一条线段连接。
5. 重复步骤3和步骤4， 但是总忽略已经击出的点。

大体上， 这个过程类似于计算凸包的卷包裹算法， 但是不同在于其循环永远不会停止。 对于一个凸包上有 h 个点的点集， 存在 $2h$ 个凸螺旋线： 对于每个起点有顺时针和逆时针螺旋线两种。

有趣的是， 一个点集的凸螺旋线和洋葱皮可以在线性时间内相互转换。 进一步的， 类似于洋葱三角剖分， 我们可以定义一个点集的子图为凸螺旋线的螺旋三角剖分。

10.22 多边形内找两个圆心使得覆盖面积最大

多边形内找两个圆心使得覆盖面积最大_每边向内收缩R得到的多边形就是两个圆心的可行域。接下来求多边形内距离最远的两点即可(一定是凸包上的两个顶点)。

10.23 判断一个圆是否在凸包内

圆心要在凸包内，圆心到凸包每条边的距离超过r

10.24 判断一个凸包是否是稳定凸包

判断一个凸包是否是稳定凸包（可以确定唯一的形状）（即每条边上至少有三个点，题目还要求每个点都是凸包上的点）

求连续的sgn(det(res.p[i - 1] - res.p[i - 2], res.p[i] - res.p[i - 2])) != 0 的个数，如果大于等于2即代表没有三个点。需要特判首尾情况。

10.25 平面图欧拉定理应用

题意：给定n个圆，求相交后把平面分成了多少份。
分析：根据欧拉定理，连通平面图满足V - E + R = 2。 其中V是顶点数，E是边数，R是面数。这题就是求面数，但是这个不一定是连通的平面图。

所以需要转换下。可以证得，如果一个图中有X个连通的平面图，那么满足，V - E + R = X+1。

然后就转换成求这个图的边数和顶点数。（一个圆上有多少个点，就有多少条边。顶点数去重即可。）

一个凸多边形的三角剖分边数是确定的。

`num = m <= 2 ? n - 1 : n * 3 - m - 3;` n为点集个数，m为凸包上的点个数。

10.26 给定地球上两点的经纬度，求出两点间球面距离

给定地球上两点的经纬度，求出两点间球面距离

设地球为直径6875的标准球体

$$dis=r\cdot arccos(cos(y1)\cdot cos(y2)\cdot cos(x2-x1)+sin(y1)\cdot sin(y2))$$

其中y是纬度，x是经度，r是地球半径 （经度纬度切不可倒转）

10.27 n个多边形的并

```
#include <cmath>
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>
#include <iostream>

typedef double flt;
const flt eps = 1e-12, inf = 1e18, PI = acos(-1.0);
inline flt sqr(flt x) {return x * x;}
inline int sgn(flt x) {return x < -eps ? -1 : (x > eps);}
inline flt fix(flt x) {return sgn(x) == 0 ? 0 : x;}

struct point {
  flt x, y;
  point(flt x = 0, flt y = 0): x(x), y(y) {}
  bool operator < (const point &rhs) const {
    return sgn(x - rhs.x) < 0 || (sgn(x - rhs.x) == 0 && sgn(y - rhs.y)
< 0);
  }
  bool operator == (const point &rhs) const {
    return sgn(x - rhs.x) == 0 && sgn(y - rhs.y) == 0;
  }
  point operator + (const point &rhs) const {
    return point(x + rhs.x, y + rhs.y);
  }
  point operator - (const point &rhs) const {
    return point(x - rhs.x, y - rhs.y);
  }
  point operator * (const flt k) const {
    return point(x * k, y * k);
  }
  point operator / (const flt k) const {
    return point(x / k, y / k);
```

```
  }
  flt dot(const point &rhs) const {
    return x * rhs.x + y * rhs.y;
  }
  flt det(const point &rhs) const {
    return x * rhs.y - y * rhs.x;
  }
  flt norm2() const {
    return x * x + y * y;
  }
  flt norm() const {
    return hypot(x, y);
  }
};

typedef std::vector<point> poly_t;
const int N = 30;
poly_t polygon[N];

std::vector<point> convex_hull(std::vector<point> u) {
  std::sort(u.begin(), u.end());
  u.erase(std::unique(u.begin(), u.end()), u.end());
  if (u.size() < 3u) return u;
  std::vector<point> ret;
  for (size_t i = 0, o = 1, m = 1; ~i; i += o) {
    while (ret.size() > m) {
      point A = ret.back() - ret[ret.size() - 2];
      point B = ret.back() - u[i];
      if (sgn(A.det(B)) < 0) break;
      ret.pop_back();
    }
    ret.push_back(u[i]);
    if (i + 1 == u.size()) m = ret.size(), o = -1;
  }
  ret.pop_back();
  return ret;
}

inline flt ratio(const point &A, const point &B, const point &O) {
  if (sgn(A.x - B.x) == 0) return (O.y - A.y) / (B.y - A.y);
  else return (O.x - A.x) / (B.x - A.x);
}

flt polygon_union(poly_t poly[], int n) {
  flt ret = 0;
  for (int i = 0; i < n; ++i) {
    for (size_t v = 0; v < poly[i].size(); ++v) {
      point A = poly[i][v], B = poly[i][(v + 1) % poly[i].size()];
      std::vector<std::pair<flt, int> > segs;
      segs.push_back(std::make_pair(0.0, 0));
      segs.push_back(std::make_pair(1.0, 0));
      for (int j = 0; j < n; ++j) if (i != j) {
        for (size_t u = 0; u < poly[j].size(); ++u) {
          point C = poly[j][u], D = poly[j][(u + 1) % poly[j].size()];
```

```
          int sc = sgn((B - A).det(C - A)), sd = sgn((B - A).det(D -
A));
          if (sc == 0 && sd == 0) {
            if (sgn((B - A).dot(D - C)) > 0 && i > j) {
              segs.push_back(std::make_pair(ratio(A, B, C), +1));
              segs.push_back(std::make_pair(ratio(A, B, D), -1));
            }
          } else {
            flt sa = (D - C).det(A - C), sb = (D - C).det(B - C);
            if (sc >= 0 && sd < 0) segs.push_back(std::make_pair(sa /
(sa - sb), 1));
            else if (sc < 0 && sd >= 0) segs.push_back(std::make_pair(sa
/ (sa - sb), -1));
          }
        }
      }
      std::sort(segs.begin(), segs.end());
      flt pre = std::min(std::max(segs[0].first, 0.0), 1.0), now, sum =
0;
      int cnt = segs[0].second;
      for (size_t j = 1; j < segs.size(); ++j) {
        now = std::min(std::max(segs[j].first, 0.0), 1.0);
        if (!cnt) sum += now - pre;
        cnt += segs[j].second;
        pre = now;
      }
      ret += A.det(B) * sum;
    }
  }
  return ret / 2;
}

void run(int cas) {
  printf("Case %d: ", cas);
  flt h, x, y, th, ret = 0;
  int n;
  std::cin >> h >> x >> y >> n;
  for (int i = 0; i < n; ++i) {
    point rect[4];
    for (int j = 0; j < 3; ++j) {
      std::cin >> rect[j].x >> rect[j].y;
    }
    std::cin >> th;
    for (int j = 0; j < 3; ++j) {
      if (sgn((rect[(j + 1) % 3] - rect[j]).dot(rect[(j + 2) % 3] -
rect[j])) == 0) {
        std::swap(rect[j], rect[1]);
        rect[3] = rect[0] + rect[2] - rect[1];
        break;
      }
    }
    polygon[i].clear();
    for (int j = 0; j < 4; ++j) {
      polygon[i].push_back(rect[j]);
```

```
      polygon[i].push_back(point(rect[j].x + x * th / h, rect[j].y + y *
th / h));
    }
    polygon[i] = convex_hull(polygon[i]);
    ret -= fabs((rect[0] - rect[1]).det(rect[2] - rect[1]));     // 减掉
阴影面积
  }
  ret += polygon_union(polygon, n);
  printf("%.3f\n", ret);
}

int main() {
  int T;
  scanf("%d", &T);
  for (int cas = 1; cas <= T; ++cas) {
    run(cas);
  }
  return 0;
}
/*
Description

The local government has just passed a motion that more trees should be
planted in down town. But as it is down town, there are many high
buildings and skyscrapers crowded in the region. They would not only
limit the area for planting, but, maybe more seriously, block the
sunlight which causes the tree hard to survive. So the government must
measure the shadow area on the ground which caused by these high
buildings precisely before they take the next step.

Buildings are rectangular blocks. And you are given a sunlight direction
and the location of all the buildings in the region. Your job is to
calculate the exact shadow area on the ground.

Input

The first line of input contains one integer, T (T ≤ 20), the number of
test cases. Each case starts with a separate line containing three
floating number, H, X, Y(0<H≤200, −200≤X,Y≤200), which specify the
direction of sunlight, that is, if there is a object at (0,0,H), its
shadow on the ground will be located at (X, Y, 0). The next line of the
case contains a single integer, N (N≤20), the number of high buildings.
The following N lines describe these N buildings separately. The i-th
line consists of seven floating numbers, xi1, yi1, xi2, yi2, xi3, yi3
and hi(-2,000,000≤ xi1, yi1, xi2, yi2, xi3, yi3≤ 2,000,000, 0<hi≤200),
that is, (xi1, yi1, 0), (xi2, yi2, 0), (xi3, yi3, 0) is three corner of
the rectangular base of the building and hi stands for the height of the
building.

Output

For each test case, print one line in the following format,
Case t: S
```

```
where t stands for the case id (1-based) and S is a floating number with
three digits after decimal point which stands for the total shadow area
on the ground.

Sample Input

2
1 2 3
1
0 0 2 0 2 1 10
1 2 3
2
0 0 2 0 2 1 10
0 2 2 2 2 3 4
Sample Output

Case 1: 80.000
Case 2: 95.000

*/
```

10.28 模拟退火伪代码

```
/* 2:  *  J(y)：在状态y时的评价函数值
   3:  *  Y(i)：表示当前状态
   4:  *  Y(i+1)：表示新的状态
   5:  *  r：  用于控制降温的快慢
   6:  *  T：  系统的温度，系统初始应该要处于一个高温的状态
   7:  *  T_min ：温度的下限，若温度T达到T_min，则停止搜索
   8: */

   9: while( T > T_min )
  10: {
  11:     dE = J( Y(i+1) ) - J( Y(i) ) ;
  13:     if ( dE >= 0 )  //表达移动后得到更优解，则总是接受移动
  14:          Y(i+1) = Y(i) ;  //接受从Y(i)到Y(i+1)的移动
  15:     else
  16:     {
  17:      // 函数exp( dE/T )的取值范围是(0,1) ，dE/T越大，则exp( dE/T )也
  18:          if ( exp( dE/T ) > random( 0 , 1 ) )
  19:              Y(i+1) = Y(i) ;  //接受从Y(i)到Y(i+1)的移动
  20:     }
  21:     T = r * T ;  //降温退火 ，0<r<1 。r越大，降温越慢；r越小，降温越快
  22:     /*
  23:     若r过大，则搜索到全局最优解的可能会较高，但搜索的过程也就较长。若r过小，则
搜索的过程会很快，但最终可能会达到一个局部最优值
  24:     */
  25:     i ++ ;
  26: }
```

11.1 旋转卡壳汇总

11.2 凸包直径(平面最远点对，旋转卡壳)

```
// 使用凸包旋转卡壳算法，传入一个凸包，输出凸包上最远欧几里得距离，&First, &Second 最远两个点的对应标号 O(n)
double convex_diameter(polygon_convex &a, int &first, int &second)
{
    vector<point> &p = a.p;
    int n = p.size();
    double maxd = 0.0;
    if(n == 1){
        first = second = 0;
        return maxd;
    }
#define next(i) ((i + 1) % n)
    for(int i = 0, j = 1; i < n; i ++){
        while(sgn(det(p[next(i)] - p[i], p[j] - p[i]) - det(p[next(i)] - p[i], p[next(j)] - p[i])) < 0){
            j = next(j);
        }
        double d = dist(p[i], p[j]);
        if(d > maxd){
            maxd = d;
            first = i, second = j;
        }
        d = dist(p[next(i)], p[next(j)]);
        if(d > maxd){
            maxd = d;
            first = i, second = j;
        }
    }
    return maxd;
}
```

11.3 最小矩形覆盖面积(旋转卡壳)

```
const LL INF = 1e18;

int sgn(double x)
{
    if(fabs(x) < eps) return 0;
    if(x > 0) return 1;
    return -1;
}

struct point
{
    double x, y;
    bool operator < (const point &tmp) const{
        if(sgn(x - tmp.x) == 0) return sgn(y - tmp.y) < 0;
        return sgn(x - tmp.x) < 0;
```

```
    }
    bool operator == (const point &tmp) const{
        return sgn(x - tmp.x) == 0 && sgn(y - tmp.y) == 0;
    }
}p[N], st[N];

double cross(point a, point b, point c)
{
    return (c.x - a.x) * (b.y - a.y) - (b.x - a.x) * (c.y - a.y);
}
double dot(point a, point b, point c)
{
    double s1 = b.x - a.x;
    double t1 = b.y - a.y;
    double s2 = c.x - a.x;
    double t2 = c.y - a.y;
    return s1 * s2 + t1 * t2;
}

int ConvexHull(point *p, int n, point *st)
{
    sort(p, p + n);
    n = unique(p, p + n) - p;
    int m = 0;
    for(int i = 0; i < n; i ++){
        while(m > 1 && cross(st[m - 2], p[i], st[m - 1]) <= 0) m --;
        st[m ++] = p[i];
    }
    int k = m;
    for(int i = n - 2; i >= 0; i --){
        while (m > k && cross(st[m - 2], p[i], st[m - 1]) <= 0) m --;
        st[m ++] = p[i];
    }
    if(n > 1) m --;
    return m;
}

double dist(point a, point b)
{
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

double rotating_calipers(point *p, int n)
{
    int R = 1, U = 1, L;
    double ans = 1e18;
    p[n] = p[0];
    for(int i = 0; i < n; i ++){
        while(sgn(cross(p[i], p[i + 1], p[U + 1]) - cross(p[i], p[i +
1], p[U])) <= 0) U = (U + 1) % n;
        while(sgn(dot(p[i], p[i + 1], p[R + 1]) - dot(p[i], p[i + 1],
p[R])) > 0) R = (R + 1) % n;
        if(i == 0) L = R;
        while(sgn(dot(p[i], p[i + 1], p[L + 1]) - dot(p[i], p[i + 1],
p[L])) <= 0) L = (L + 1) % n;
```

```
        double d = dist(p[i], p[i + 1]) * dist(p[i], p[i + 1]);
        double area = fabs(cross(p[i], p[i + 1], p[U])) * fabs(dot(p[i],
p[i + 1], p[R]) - dot(p[i], p[i + 1], p[L])) / d;
        if(area < ans) ans = area;
    }
    return ans;
}

int main()
{
    int n;
    scanf("%d", &casenum);
    for(casei = 1; casei <= casenum; casei ++){
        scanf("%d", &n);
        n *= 4;
        for(int i = 0; i < n; i ++){
            scanf("%lf%lf", &p[i].x, &p[i].y);
        }
        int m = ConvexHull(p, n, st);
        double ans;
        if(m < 3) ans = 0;
        else ans = rotating_calipers(st, m);
        printf("Case #%d:\n%.0lf\n", casei, ans);
    }
    return 0;
}
```

11.4 求两凸包的最近点对距离(旋转卡壳)

```
const int N = 50000;
const double eps = 1e-9;
const double INF = 1e99;

struct Point
{
    double x, y;
};

Point P[N], Q[N];

double cross(Point A, Point B, Point C)
{
    return (B.x - A.x) * (C.y - A.y) - (B.y - A.y) * (C.x - A.x);
}

double dist(Point A, Point B)
{
    return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
}

double multi(Point A, Point B, Point C)
{
    return (B.x - A.x) * (C.x - A.x) + (B.y - A.y) * (C.y - A.y);
}
```

```
//顺时针排序
void anticlockwise(Point p[], int n)
{
    for(int i = 0; i < n - 2; i ++){
        double tmp = cross(p[i], p[i+1], p[i+2]);
        if(tmp > eps) return;
        else if(tmp < -eps){
            reverse(p, p + n);
            return;
        }
    }
}

//计算C点到直线AB的最短距离
double Getdist(Point A, Point B, Point C)
{
    if(dist(A, B) < eps) return dist(B, C);
    if(multi(A, B, C) < -eps) return dist(A, C);
    if(multi(B, A, C) < -eps) return dist(B, C);
    return fabs(cross(A, B, C) / dist(A, B));
}

//求一条直线的两端点到另外一条直线的距离，反过来一样，共4种情况
double MinDist(Point A, Point B, Point C, Point D)
{
    return min(min(Getdist(A, B, C), Getdist(A, B, D)), min(Getdist(C,
D, A), Getdist(C, D, B)));
}

double Solve(Point P[], Point Q[], int n, int m)
{
    int yminP = 0, ymaxQ = 0;
    for(int i = 0; i < n; i ++)
        if(P[i].y < P[yminP].y) yminP = i;
    for(int i = 0; i < m; i ++)
        if(Q[i].y > Q[ymaxQ].y) ymaxQ = i;
    P[n] = P[0];
    Q[m] = Q[0];
    double tmp, ans = INF;
    for(int i = 0; i < n; i ++){
        while(tmp = cross(P[yminP + 1], Q[ymaxQ + 1], P[yminP]) -
cross(P[yminP + 1], Q[ymaxQ], P[yminP]) > eps)
            ymaxQ = (ymaxQ + 1) % m;
        if(tmp < -eps) ans = min(ans, Getdist(P[yminP], P[yminP+1],
Q[ymaxQ]));
        else ans = min(ans, MinDist(P[yminP], P[yminP+1], Q[ymaxQ],
Q[ymaxQ+1]));
        yminP = (yminP + 1) % n;
    }
    return ans;
}

int main()
{
    int n,m;
```

```
    while(cin >> n >> m)
    {
        if(n == 0 && m == 0) break;
        for(int i = 0; i < n; i ++)
            cin >> P[i].x >> P[i].y;
        for(int i = 0; i < m; i ++)
            cin >> Q[i].x >> Q[i].y;
        anticlockwise(P, n);
        anticlockwise(Q, m);
        printf("%.5lf\n", min(Solve(P, Q, n, m), Solve(Q, P, m, n)));
    }
    return 0;
}
```

11.5 求平面内面积最大的三角形（四边形）

hdu2202(bzoj1069求面积最大的四边形(先枚举对角线即可))
// 旋转卡壳求平面内面积最大的三角形(先求出凸包再枚举凸包上的两个点(不能只是相邻点，因为可能并不是最大))（可以求出凸包内最远两点距离及凸包宽度）

三角形面积公式：
1.s=a*h/2 a:底 h:高
2.s=l*r/2 l:三角形周长 r:内切圆半径
3.s=l/(4*r) l:三角形三边长乘积 r:外接圆半径
4.s=sqrt(p*(p-a)*(p-b)*(p-c)) a,b,c 三边长
5.s=1/2*a*b*sin(c) a,b两边长,c两边夹角
6.s=1/2*abs(x1*y2+x2*y3+x3*y1-x1*y3-x2*y1-x3*y2);

```
// 二维平面上的三角形面积公式
double TriangleArea(point a, point b, point c)
{
    return 0.5 * fabs(a.x * b.y + b.x * c.y + c.x * a.y - a.x * c.y -
b.x * a.y - c.x * b.y);
}
// 平面三角形的有向面积(需要再*0.5)
double cross(point a, point b, point c)
{
    double x1 = b.x - a.x;
    double x2 = c.x - a.x;
    double y1 = b.y - a.y;
    double y2 = c.y - a.y;
    return x1 * y2 - x2 * y1;
}
double dot(point A, point B, point C)
{
    return (B.x - A.x) * (C.x - A.x) + (B.y - A.y) * (C.y - A.y);
}
// 注意通常都有一句p[n] = p[0]; 这里的点需要先求出凸包（逆时针）
for(int i = 0; i < m; i ++){
    int q = 1;   // int L = (i + 3) % m, R = (i + 1) % m;
```

```
    for(int j = i + 1(四边形是i + 2); j < m; j ++){ // 这里是枚举第二个点，和第一个点连成边
        //while(det(a.p[j] - a.p[i], a.p[q + 1] - a.p[i])> det(a.p[j] - a.p[i], a.p[q] - a.p[i]))
        //    q = (q + 1) % m;
        //gmax(ans, 0.5 * det(a.p[j] - a.p[i], a.p[q] - a.p[i]));
        while(cross(a.p[i], a.p[j], a.p[q + 1]) > cross(a.p[i], a.p[j], a.p[q])) q = (q + 1) % m;
     /* 特别的，这里是求四边形的方法。 这里的cross比较，事实上只是比较角度罢了，角度越大越靠外。 终于明白啦。
     //while((L + 1) % m != i && cross(a.p[i], a.p[j], a.p[L + 1]) > cross(a.p[i], a.p[j], a.p[L])) L = (L + 1) % m;
        //while((R + 1) % m != j && cross(a.p[j], a.p[i], a.p[R + 1]) > cross(a.p[j], a.p[i], a.p[R])) R = (R + 1) % m;
     //gmax(ans, 0.5 * cross(a.p[i], a.p[j], a.p[L]) + 0.5 * cross(a.p[j], a.p[i], a.p[R]));
     */
        gmax(ans, 0.5 * cross(a.p[i], a.p[j], a.p[q]));

    }
}
```

11.6 求最小面积外接矩形

hdu5251(bzoj1185需要输出矩形四点坐标)

```
// 旋转卡壳求最小面积外接矩形
// 对于每一个旋转卡壳都要注意对应的cross的方向
double cross(point a, point b, point c)
{
    return (c.x - a.x) * (b.y - a.y) - (b.x - a.x) * (c.y - a.y);
}
// 将点c沿直线ab法向量方向平移距离len得到的直线
point move_d(point a, point b, point c, const double &len)
{
    point d = b - a;
    d = d / d.norm();
    d = rotate_point(d, PI / 2);
    return point(c + d * len);
}
double rotating_calipers(point *p, int n)
{
    //首先求一个凸包将这些点给覆盖住，就转换成求一个面积最小的矩形将凸包覆盖住。很明显矩形的一条边肯定是凸包的一条边。因此我们枚举凸包的边，然后旋转卡壳求最左边，最右边，最上边的那三个点就可以确定这个矩形了。剩下的就是用几何知识来求这个矩形的面积了。
    //根据最上面的那个点与我们枚举的那个边，通过叉积可以确定矩形的高，然后根据最左边的点与最右边的点和我们枚举的边可以确定矩形的底，知道高和底就可以求矩形的面积了。(其他方法可自想)
    int R = 1, U = 1, L = 0;
    double ans = 1e18;
    p[n] = p[0];
```

```
    for(int i = 0; i < n; i ++){
        while(sgn(cross(p[i], p[i + 1], p[U + 1]) - cross(p[i], p[i +
1], p[U])) <= 0) U = (U + 1) % n;// 差积查询对踵点
        while(sgn(dot(p[i], p[i + 1], p[R + 1]) - dot(p[i], p[i + 1],
p[R])) > 0) R = (R + 1) % n;// 点积表示面积，查询左右两个点
        if(i == 0) L = R;
        while(sgn(dot(p[i], p[i + 1], p[L + 1]) - dot(p[i], p[i + 1],
p[L])) <= 0) L = (L + 1) % n;
        double d = dist(p[i], p[i + 1]) * dist(p[i], p[i + 1]);
        double area = fabs(cross(p[i], p[i + 1], p[U])) * fabs(dot(p[i],
p[i + 1], p[R]) - dot(p[i], p[i + 1], p[L])) / d;
        //double area;
        if(area < ans) {
           ans = area;
           // 特别地， 这里是求矩形的四个顶点坐标
           LD = PointProjLine(p[L], p[i], p[i + 1]);
            RD = PointProjLine(p[R], p[i], p[i + 1]);
            if(!(LD == p[L])){
                LU = PointProjLine(p[U], p[L], LD);
            }
            else{
                point PL;
                PL = move_d(p[i], p[i + 1], LD, 1);
                LU = PointProjLine(p[U], p[L], PL);
            }
            if(!(RD == p[R])){
                RU = PointProjLine(p[U], p[R], RD);
            }
            else{
                point PR;
                PR = move_d(p[i], p[i + 1], RD, 1);
                RU = PointProjLine(p[U], p[R], PR);
            }
     }
    }
    return ans;
}
```

11.7 求两个凸包间的最短距离

hdu2823

```
double cross(point A, point B, point C)
{
    return (B.x - A.x) * (C.y - A.y) - (B.y - A.y) * (C.x - A.x);
}
```

// 求两个凸包间的最短距离，判断一个凸包是否在另一个凸包内，不能只判断一个凸多边形的点或边是否在另一个凸多边形内，而应该用半平面求交，半平面求交时需要注意求出的是一个点的情况。 头文件complex.h不行则用complex

```
void anticlockwise(point p[], int n)
{
    for(int i = 0; i < n - 2; i ++){
        double tmp = cross(p[i], p[i+1], p[i+2]);
        if(tmp > eps) return;
```

```
        else if(tmp < -eps){
            reverse(p, p + n);
            return;
        }
    }
}
//计算C点到直线AB的最短距离
double Getdist(point A, point B, point C)
{
    if(dist(A, B) < eps) return dist(B, C);
    if(dot(A, B, C) < -eps) return dist(A, C);
    if(dot(B, A, C) < -eps) return dist(B, C);
    return fabs(cross(A, B, C) / dist(A, B));
}

//求一条直线的两端点到另外一条直线的距离，反过来一样，共4种情况
double MinDist(point A, point B, point C, point D)
{
    return min(min(Getdist(A, B, C), Getdist(A, B, D)), min(Getdist(C,
D, A), Getdist(C, D, B)));
}

double Solve(point P[], point Q[], int n, int m)
{
    int yminP = 0, ymaxQ = 0;
    // 下面这两句似乎不写也能ac
    for(int i = 0; i < n; i ++) if(P[i].y < P[yminP].y) yminP = i;
    for(int i = 0; i < m; i ++) if(Q[i].y > Q[ymaxQ].y) ymaxQ = i;
    P[n] = P[0];
    Q[m] = Q[0];
    double ans = INF;
    for(int i = 0; i < n; i ++){
        while(cross(P[yminP + 1], Q[ymaxQ + 1], P[yminP]) -
cross(P[yminP + 1], Q[ymaxQ], P[yminP]) > eps)
            ymaxQ = (ymaxQ + 1) % m;
        ans = min(ans, MinDist(P[yminP], P[yminP+1], Q[ymaxQ],
Q[ymaxQ+1]));
        yminP = (yminP + 1) % n;
    }
    return ans;
}
int main()
{
    anticlockwise(P, n);
    anticlockwise(Q, m);
    // 这句似乎只求一个也可
    printf("%.4lf\n", min(Solve(P, Q, n, m), Solve(Q, P, m, n)));
}
```

12.1 其他知识或技巧

poj1755 给出n个人铁人三项的速度，问我们是否可以分别改变每个项目的路程，使得第i个人严格获胜。(半平面交_模型转换)

假设我们已知其中两个人的三项速度分别为a1,a2,a3,b1,b2,b3, 每个项目的路程分别为x,y,z

要使第一个人获胜, 那么有:

```
x / a1 + y / a2 + z / a3 < x / b1 + y / b2 + z / b3;
=>  b1b2b3(a2a3x + a1a3y + a1a2z) < a1a2a3(b2b3x + b1b3y + b1b2y)
=>  a2a3b2b3(b1-a1)x + a1a3b1b3(b2-a2)y + a1a2b1b2(b3-a3)z < 0
=>  a2a3b2b3(b1-a1)x/z + a1a3b1b3(b2-a2)y/z + a1a2b1b2(b3-a3)z/z < 0
=>  a2a3b2b3(b1-a1)x/z + a1a3b1b3(b2-a2)y/z + a1a2b1b2(b3-a3) < 0
```

然后将x/z和y/z看做X和Y，原来的系数看做A,B,C，就成了下面的形式

A·X+B·Y+C<0

这时候我们就可以用半平面交来求解。

atan2(y, x) 返回-PI ~ PI;

题意:

给定一个点P，求椭圆和三角形OPA相交的面积。

类型:

公式法||积分法

分析:

公式法:

假设椭圆上点M(x, y), N(x, -y), x, y > 0 A(a, 0), 原点(0, 0) 则有

S(OAM) = 0.5 * a * b * arccos(x / a), S(MAN) = a * b * arccos(x / a) - x * y;

利用k设定一下椭圆交点和给定x点的比例即可求出OP和椭圆上的交点。（由题意可得点一定不在椭圆内）

积分法:

求三角形和去掉三角形的奇怪形状的面积和。

```
int main()
{
    scanf("%d", &casenum);
    for(casei = 1; casei <= casenum; casei ++){
        scanf("%lf%lf%lf%lf", &a, &b, &x, &y);
        double k = sqrt((a * a * b * b) / (b * b * x * x + a * a * y * y));
        printf("%.2lf\n", 0.5 * a * b * acos(k * x / a));
    }
    return 0;
}
```