计算几何

目录

直线求交点

```
Point getSol(Point a, Point b, Point c, Point d) {//注意平行！
    double s = (d.y - b.y) * (b.x - a.x) * (d.x - c.x) + (b.y - a.y)
* b.x
            * (d.x - c.x) - (d.y - c.y) * (b.x - a.x) * d.x;
    double t = (b.y - a.y) * (d.x - c.x) - (d.y - c.y) * (b.x - a.x);
    double x = s / t;
    double s1 = (d.x - b.x) * (b.y - a.y) * (d.y - c.y) + (b.x - a.x)
* b.y
            * (d.y - c.y) - (d.x - c.x) * (b.y - a.y) * d.y;
    double t1 = (b.x - a.x) * (d.y - c.y) - (d.x - c.x) * (b.y -
a.y);
    double y = s1 / t1;
    Point res(x, y);
    return res;
}
```

判定线段相交

```
int inSeg(const Point& a, const Point& b, const Point& x) {
    int p = (b - a).dot(x - a);
    int q = (a - b).dot(x - b);
    if (p >= 0 && q >= 0)
        return 1;
    return 0;
}
int isCross(const Point& a, const Point& b, const Point& c, const
Point& d) {
    int p = (b - a).cross(c - a);
    int q = (b - a).cross(d - a);
    int r = (d - c).cross(a - c);
    int s = (d - c).cross(b - c);
    if (p * q < 0 && r * s < 0)
        return 1;
    else if (c == a || c == b || d == a || d == b)
        return 1;
    else if (p * q <= 0 && r * s <= 0 && (p * q == 0 || r * s == 0))
{
        if (inSeg(a, b, d) || inSeg(a, b, d) || inSeg(c, d, a)
                || inSeg(c, d, b))
            return 1;
    }
    return 0;
}
```

旋转平移伸缩

```
complex<double> ei(double theta) {
    complex<double> r(0, theta);
    return exp(r);
}
Point trans(const Point& p, double theta, double fac = 1.0) {
    complex<double> s(p.x, p.y);
    s = s * ei(theta) * fac;
    Point res(s.real(), s.imag());
    return res;
}
```

极角排序求凸包

```
const int maxn = 10010;
struct Point {
    double x, y;
    int id;
    bool operator<(const Point& p) const {
        const Point zero(0, 0);
        double t = this->cross(p);
        if (sig(t))
            return sig(t) > 0;
        return sig(this->dist(zero) - p.dist(zero)) > 0;
    }
};
Point point[maxn], stack[maxn];
int N, top;
void convexHull() {
    for (int i = 1; i <= N; i++) {
        if (sig(point[1].y - point[i].y) > 0 || (sig(point[1].y -
point[i].y)
                == 0 && sig(point[1].x - point[i].x) > 0))
            swap(point[1], point[i]);
    }
    for (int i = 2; i <= N; i++)
        point[i] = point[i] - point[1];
    sort(point + 2, point + 1 + N);
    int n = N;
    N = 1;
    const Point zero(0, 0);
    point[1] = zero;
    for (int i = 2; i <= n;) {
        Point current = point[i];
        point[++N] = current;
        for (; i <= n && !sig(current.cross(point[i])); i++)
            ;
    }
    top = 0;
    stack[top++] = point[1];
    stack[top++] = point[2];
    for (int i = 3; i <= N; i++) {
        while (top >= 2 && sig((stack[top - 1] - stack[top -
2]).cross(point[i]
                - stack[top - 2])) < 0)
            top--;
        stack[top++] = point[i];
    }
}
double L;
void init() {
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf", &point[i].x, &point[i].y);
    }
}
void work() {
    convexHull();
    stack[top] = stack[0];
    double c = 0;
    for (int i = 0; i < top; i++)
        c += stack[i].dist(stack[i + 1]);
    c += 2 * PI * L;
    printf("%.0lf\n", c);
}
```

半平面求交

```
const double eps = 1e-8;
const int maxn = 50010;
const double INF = 1e5;
int sig(double t) {
    return (t > eps) - (t < -eps);
}
struct Line {
    Point a, b;
    double degree;

    Line() {}
    Line(const Point& p, const Point& q) {a = p;b = q;}
    Line(double p, double q, double r, double s) {
        a.x = p;
        a.y = q;
        b.x = r;
        b.y = s;
    }
    inline void build() {
        degree = atan2(b.y - a.y, b.x - a.x);
    }
    inline bool operator<(const Line& p) const {
        if (sig(degree - p.degree))
            return sig(degree - p.degree) < 0;
        double t = (b - a).cross(p.b - a);
        return sig(t) < 0;
    }
    Point intersect(const Line& p) {
        double rx, ry;
        getSol(a, b, p.a, p.b, rx, ry);
        Point res(rx, ry);
        return res;
    }
    inline bool operator==(const Line& p) {
        return a == p.a && b == p.b;
    }
};
int judge(Line& p, Line& q, Line& r) {
    Point pnt = p.intersect(q);
    if (sig((r.b - r.a).cross(pnt - r.a)) < 0)
        return 1;
    return 0;
}
Line queue[5 * maxn], line[maxn], result[maxn];
Point resultP[maxn];
int R, L, Rp;
void halfPlaneIntersection() {
    R = 0;
    Rp = 0;
    sort(line + 1, line + 1 + L);
    int head = 0, rear = 0;
    int l = L;
    L = 0;
    for (int i = 1; i <= l;) {
        Line current = line[i];
        line[++L] = current;
        for (; i <= l && sig(current.degree - line[i].degree) == 0; i+
+)
            ;
    }
    queue[rear++] = line[1];
    queue[rear++] = line[2];
    for (int i = 3; i <= L; i++) {
        while (rear - head >= 2 && judge(queue[rear - 2], queue[rear -
1],
                line[i]))
            rear--;
        while (rear - head >= 2 && judge(queue[head + 1], queue[head],
line[i]))
            head++;
        queue[rear++] = line[i];
    }
    int flag = 1;
    while (flag) {
        flag = 0;
        while (rear - head >= 2 && judge(queue[rear - 2], queue[rear -
1],
                queue[head])) {
            rear--;
            flag = 1;
        }
        while (rear - head >= 2 && judge(queue[head + 1], queue[head],
                queue[rear - 1])) {
            head++;
            flag = 1;
        }
    }
    queue[rear++] = queue[head];
    for (int i = head; i < rear - 1; i++) {
        resultP[++Rp] = queue[i].intersect(queue[i + 1]);
    }
    for (int i = head; i < rear; i++) {
        result[++R] = queue[i];
    }
}
Line up(INF, INF, -INF, INF), down(-INF, -INF, INF, -INF), left(-INF,
INF,
        -INF, -INF), right(INF, -INF, INF, INF);
Point points[maxn];
int N;
void init() {
    L = 0;
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf", &points[i].x, &points[i].y);
    }
    points[N + 1] = points[1];
    points[N + 2] = points[2];
    int isRight = 0;
    for (int i = 3; i <= N + 2; i++) {
        if (sig(
                (points[i - 1] - points[i - 2]).cross(points[i] -
points[i - 1]))
                < 0) {
            isRight = 1;
            break;
        }
    }
    if (isRight) {
        for (int i = N - 1; i >= 1; i--) {
            line[++L].a = points[i + 1];
            line[L].b = points[i];
        }
        line[++L].a = points[1];
        line[L].b = points[N];
    } else {
        for (int i = 1; i <= N - 1; i++) {
            line[++L].a = points[i];
            line[L].b = points[i + 1];
        }
        line[++L].a = points[N];
        line[L].b = points[1];
    }
    line[++L] = up;
    line[++L] = down;
    line[++L] = left;
    line[++L] = right;
    for (int i = 1; i <= L; i++)
        line[i].build();
}
void work() {
    halfPlaneIntersection();
    if (R <= 3) {
        puts("Surveillance is impossible.");
        return;
    }
    for (int i = 1; i <= R; i++) {
        if (result[i] == up || result[i] == down || result[i] == left
                || result[i] == right) {
            puts("Surveillance is impossible.");
            return;
        }
    }
    puts("Surveillance is possible.");
}
```

光线圆面交

```
struct Line {
    Point start;
    Point dir;
};
struct Circle {
    Point center;
    double r;

    int isin(const Point& p) const {
        return sig(p.dist(center) - r) < 0;
    }
};
Circle circle[30];
Line current;
int N;
void init() {
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf%lf", &circle[i].center.x, &circle[i].center.y,
                &circle[i].r);
    }
    scanf("%lf%lf%lf%lf", &current.start.x, &current.start.y,
&current.dir.x,
            &current.dir.y);
}
Point getDir(const Line& source, const Point& p, const Point& q,
double theta) {
    double t = (q - p).cross(source.dir);
    if (sig(t) > 0) {
        Point res = rotate(current.dir, -2 * theta);
        return res;
    } else {
        Point res = rotate(current.dir, 2 * theta);
        return res;
    }
}
int getIntersect() {
    Point A = current.start;
    Point B = current.start + current.dir;
    Line next;
    int idx = -1;
    double res = -1;
    for (int i = 1; i <= N; i++) {
        Point C = circle[i].center;
        Point D = rotate(current.dir, PI / 2) + C;
        Point E = getSol(A, B, C, D);
        if (!circle[i].isin(E))
            continue;
        double dis = C.dist(E);
        double theta = fixacos(dis / circle[i].r);
        double PE = sqrt((circle[i].r * circle[i].r) - (dis * dis));
        Point P = current.start + rotate(E - current.start, 0,
(E.dist(
                current.start) - PE) / E.dist(current.start));
        Point Q = current.start + rotate(E - current.start, 0,
(E.dist(
                current.start) + PE) / E.dist(current.start));
        if (sig(current.dir.dot(P - current.start)) < 0 ||
sig(current.dir.dot(
                Q - current.start)) < 0)
            continue;
        double t = P.dist(current.start);
        double s = Q.dist(current.start);
        if (sig(t - s) < 0) {
            if (res == -1 || sig(res - t) > 0) {
                res = t;
                idx = i;
                next.start = P;
                next.dir = getDir(current, C, E, theta);
            }
        } else {
            if (res == -1 || sig(res - s) > 0) {
                res = s;
                idx = i;
                next.start = Q;
                next.dir = getDir(current, C, E, theta);
            }
        }
    }
    if (idx != -1)
        current = next;
    return idx;
}
int result[100], R;
void work() {
    memset(result, 0, sizeof(result));
    R = 0;
    for (int i = 1; i <= 11; i++) {
        int idx = getIntersect();
        if (idx == -1) {
            result[R++] = -1;
            break;
        }
        result[R++] = idx;
    }
    if (result[10] != -1)
        result[10] = -2;
    for (int i = 0; i < R; i++) {
        if (result[i] > 0)
            printf("%d", result[i]);
        else if (result[i] == -1)
            printf("inf");
        else if (result[i] == -2)
            printf("...");
        if (i == R - 1)
            puts("");
        else
            putchar(' ');
    }
    puts("");
}
```

弧相交

```
Point result[100000];
int C, N;
struct Arc {
    Point center;
    double R;
    Point A, B, M;
    void build() {
        double stheta = (A - M).cross(B - M) / M.dist(A) / M.dist(B);
        R = fabs(0.5 * A.dist(B) / stheta);
        Point M1 = (A + M);
        M1.x /= 2;
        M1.y /= 2;
        Point M2 = (B + M);
        M2.x /= 2;
        M2.y /= 2;
        Point P1 = trans(M - M1, PI / 2, 1) + M1;
        Point P2 = trans(M - M2, PI / 2, 1) + M2;
        center = getSol(P1, M1, P2, M2);
        if (A < B)
            swap(A, B);
    }
    int isin(const Point& p) const {
        double t = (p - M).cross(A - M);
        double s = (p - M).cross(B - M);
        if (sig(t * s) >= 0)
            return 1;
        return 0;
    }
    int isInf(const Arc& p) const {
        if (center == p.center && sig(R - p.R) == 0) {
            if (isin(p.A) && !(p.A == A) && !(p.A == B))
                return 1;
            if (isin(p.B) && !(p.B == A) && !(p.B == B))
                return 1;
            if (p.isin(A) && !(A == p.A) && !(A == p.B))
                return 1;
            if (p.isin(B) && !(B == p.A) && !(B == p.B))
                return 1;
            if (A == p.A && B == p.B && isin(p.M))
                return 1;
        }
        return 0;
    }
    void intersect(const Arc& p) const {
        double d = center.dist(p.center);
        if (sig(d - R - p.R) > 0 || sig(d - fabs(R - p.R)) < 0)
            return;
        if (center == p.center && sig(R - p.R) == 0) {
            if (p.A == A || p.A == B)
                result[C++] = p.A;
            if (p.B == A || p.B == B)
                result[C++] = p.B;
            return;
        }
        double theta = fixacos((R * R + d * d - p.R * p.R) / 2 / R /
d);
        Point t = trans(p.center - center, theta, R / d) + center;
        if (isin(t) && p.isin(t))
            result[C++] = t;
        Point s = trans(p.center - center, -theta, R / d) + center;
        if (isin(s) && p.isin(s))
            result[C++] = s;
    }
};
Arc arc[110];
void init() {
    scanf("%d", &N);
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf", &arc[i].A.x, &arc[i].A.y);
        scanf("%lf%lf", &arc[i].B.x, &arc[i].B.y);
        scanf("%lf%lf", &arc[i].M.x, &arc[i].M.y);
        arc[i].build();
    }
}
void work() {
    for (int i = 1; i <= N; i++) {
        for (int j = i + 1; j <= N; j++) {
            if (arc[i].isInf(arc[j])) {
                puts("Infinity");
                return;
            }
        }
    }
    for (int i = 1; i <= N; i++) {
        for (int j = i + 1; j <= N; j++) {
            arc[i].intersect(arc[j]);
        }
    }
    sort(result, result + C);
    int r = C;
    C = 0;
    for (int i = 0; i < r;) {
        Point current = result[i];
        result[C++] = current;
        for (; i < r && current == result[i]; i++)
            ;
    }
    printf("%d\n", C);
    for (int i = 0; i < C; i++) {
        printf("%.3lf %.3lf\n", result[i].x, result[i].y);
    }
}
```

极角排序

```
//CII 4604 统计平面坐标点集锐角三角形个数
struct Rad {
    double value;
    bool operator<(const Rad& p) const {
        return sig(value - p.value) < 0;
    }
};
int sum[1500], N, R;
Rad rad[1500];
Point point[1500];
int sumNormal(double from, double to) {
    if (from > to)
        swap(from, to);
    int low = 1, high = R, ans1 = -1;
    while (low <= high) {
        int mid = (low + high) >> 1;
        if (sig(rad[mid].value - from) >= 0) {
            ans1 = mid;
            high = mid - 1;
        } else
            low = mid + 1;
    }
    low = 1;
    high = R;
    int ans2 = -1;
    while (low <= high) {
        int mid = (low + high) >> 1;
        if (sig(rad[mid].value - to) <= 0) {
            ans2 = mid;
            low = mid + 1;
        } else
            high = mid - 1;
    }
    if (ans1 == -1 || ans2 == -1 || ans2 < ans1)
        return 0;
    return sum[ans2] - sum[ans1 - 1];
}
int getSum(double from, double to) {
    if (sig(to - PI) > 0) {
        return sumNormal(from, PI) + sumNormal(-PI, to - 2 * PI);
    } else if (sig(to + PI) < 0) {
        return sumNormal(-PI, from) + sumNormal(to + 2 * PI, PI);
    } else
        return sumNormal(from, to);
}
void init() {
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf", &point[i].x, &point[i].y);
    }
}
void work(int num) {
    if (N < 3) {
        printf("Scenario %d:\n", num);
        printf("There are 0 sites for making valid tracks\n");
        return;
    }
    int res = 0;
    for (int i = 1; i <= N; i++) {
        R = 0;
        for (int j = 1; j <= N; j++) {
            if (i == j)
                continue;
            ++R;
            Point tmp = point[j] - point[i];
            rad[R].value = atan2(tmp.y, tmp.x);
        }
        sort(rad + 1, rad + 1 + R);
        for (int j = 1; j <= R; j++)
            sum[j] = sum[j - 1] + 1;
        for (int j = 1; j <= R; j++) {
            double r = rad[j].value + PI;
            if (sig(r - PI) > 0)
                r -= 2 * PI;
            if (sig(r + PI) < 0)
                r += 2 * PI;
            res += getSum(r, r + PI / 2);
            res += getSum(r, r - PI / 2);
        }
    }
    int t = N * (N - 1) * (N - 2) / 6;
    printf("Scenario %d:\n", num);
    printf("There are %d sites for making valid tracks\n", t - res /
2);
}
```

可见圆

```
const double eps = 1e-14;
double sx, sy;
struct Point {
    double x, y;
    inline bool operator<(const Point& p) const {
        return sig(atan2(y - sy, x - sx) - atan2(p.y - sy, p.x - sx))
< 0;
    }
};
struct Circle {
    Point c;
    double r;
    int isin(const Point& p) {
        return sig(p.dist(c) - r) <= 0;
    }
};
int N;
Circle circle[110];
int result[110];
void init() {
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf%lf", &circle[i].c.x, &circle[i].c.y,
&circle[i].r);
    }
}
Point toPoint(const complex<double>& p) {
    Point res(p.real(), p.imag());
    return res;
}
int getHigh(const Point& p) {
    for (int i = N; i >= 1; i--) {
        if (circle[i].isin(p))
            return i;
    }
}
double rad[10010];
inline double getTheta(Circle& p, Circle& q) {
    if (sig(p.r * p.c.dist(q.c)) == 0)
        return PI;
    return fixacos((p.r * p.r + p.c.dist(q.c) * p.c.dist(q.c) - q.r *
q.r) / 2
            / p.r / p.c.dist(q.c));
}
int comp(const void* p, const void* q) {
    double *x = (double*) p;
    double *y = (double*) q;
    return sig(*x - *y);
}
double trim(double t) {
    if (t > PI)
        t -= PI;
    if (t < -PI)
        t += PI;
    return t;
}
void work() {
    memset(result, 0, sizeof(result));
    for (int i = 1; i <= N; i++) {
        int R = 0;
        for (int j = 1; j <= N; j++) {
            double d = circle[i].r;
            double d1 = circle[j].r;
            double d2 = circle[i].c.dist(circle[j].c);
            if (sig(d + d1 - d2) < 0 || sig(d1 + d2 - d) < 0
                    || sig(d2 + d - d1) < 0)
                continue;
            complex<double> p(circle[i].c.x, circle[i].c.y);
            complex<double> q(circle[j].c.x, circle[j].c.y);
            double theta = getTheta(circle[i], circle[j]);
            double alpha = arg(q - p);
            rad[R++] = trim(alpha + theta);
            rad[R++] = trim(alpha - theta);
        }
        qsort(rad, R, sizeof(double), comp);
        rad[R] = rad[0] + 2 * PI;
        for (int j = 0; j < R; j++) {
            double mid = (rad[j] + rad[j + 1]) / 2;
            double diff = 4e-13;
            for (int k = -1; k <= 1; k += 2) {
                complex<double> p = (circle[i].r + k * diff) * ei(mid);
                Point t = toPoint(p);
                t = t + circle[i].c;
                result[getHigh(t)] = 1;
            }
        }
    }
    int ans = 0;
    for (int i = 1; i <= N; i++) {
        if (result[i])
            ans++;
    }
    printf("%d\n", ans);
}
```

多边形与多边形面积交

```
#define maxn 510
const double eps = 1E-8;
double cross(Point o, Point a, Point b) {
    return (a.x - o.x) * (b.y - o.y) - (b.x - o.x) * (a.y - o.y);
}
double area(Point * ps, int n) {
    ps[n] = ps[0];
    double res = 0;
    for (int i = 0; i < n; i++) {
        res += ps[i].x * ps[i + 1].y - ps[i].y * ps[i + 1].x;
    }
    return res / 2.0;
}
int lineCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    s1 = cross(a, b, c);
    s2 = cross(a, b, d);
    if (sig(s1) == 0 && sig(s2) == 0)
        return 2;
    if (sig(s2 - s1) == 0)
        return 0;
    p.x = (c.x * s2 - d.x * s1) / (s2 - s1);
    p.y = (c.y * s2 - d.y * s1) / (s2 - s1);
    return 1;
}
//多边形切割
//用直线ab切割多边形p，切割后的在向量(a,b)的左侧，并原地保存切割结果
//如果退化为一个点，也会返回去，此时n为1
void polygon_cut(Point * p, int & n, Point a, Point b) {
    static Point pp[maxn];
    int m = 0;
    p[n] = p[0];
    for (int i = 0; i < n; i++) {
        if (sig(cross(a, b, p[i])) > 0)
            pp[m++] = p[i];
        if (sig(cross(a, b, p[i])) != sig(cross(a, b, p[i + 1])))
            lineCross(a, b, p[i], p[i + 1], pp[m++]);
    }
    n = 0;
    for (int i = 0; i < m; i++)
        if (!i || !(pp[i] == pp[i - 1]))
            p[n++] = pp[i];
    while (n > 1 && p[n - 1] == p[0])
        n--;
}
//返回三角形oab和三角形ocd的有向交面积，o是原点
double intersectArea(Point a, Point b, Point c, Point d) {
    Point o(0, 0);
    int s1 = sig(cross(o, a, b));
    int s2 = sig(cross(o, c, d));
    if (s1 == 0 || s2 == 0)
        return 0.0; //退化，面积为0

    if (s1 == -1)
        swap(a, b);
    if (s2 == -1)
        swap(c, d);

    Point p[10] = { o, a, b };
    int n = 3;

    polygon_cut(p, n, o, c);
    polygon_cut(p, n, c, d);
    polygon_cut(p, n, d, o);

    double res = fabs(area(p, n));
    if (s1 * s2 == -1)
        res = -res;
    return res;
}

//求两多边形的交面积
double intersectArea(Point * ps1, int n1, Point * ps2, int n2) {
    if (area(ps1, n1) < 0)
        reverse(ps1, ps1 + n1);
    if (area(ps2, n2) < 0)
        reverse(ps2, ps2 + n2);

    ps1[n1] = ps1[0];
    ps2[n2] = ps2[0];
    double res = 0;
    for (int i = 0; i < n1; i++) {
        for (int j = 0; j < n2; j++) {
            res += intersectArea(ps1[i], ps1[i + 1], ps2[j], ps2[j +
1]);
        }
    }
    return res; //assume res is positive !
}

//hdu-3060    求两个任意简单多边形的并面积
Point ps1[maxn], ps2[maxn];
int n1, n2;

int main() {
    while (scanf("%d%d", &n1, &n2) != EOF) {
        for (int i = 0; i < n1; i++)
            scanf("%lf%lf", &ps1[i].x, &ps1[i].y);
        for (int i = 0; i < n2; i++)
            scanf("%lf%lf", &ps2[i].x, &ps2[i].y);
        double ans = intersectArea(ps1, n1, ps2, n2);
        ans = fabs(area(ps1, n1)) + fabs(area(ps2, n2)) - ans; //容斥
        printf("%.2f\n", ans);
    }
    return 0;
}
```

梯形剖分

```
//统计格点多边形每个格子内的面积
struct Poly {
    Point point[10];
    int cnt;
    void init() {
        cnt = 0;
    }
    void addPoint(const Point& p) {
        point[cnt++] = p;
    }
    void finish() {
        point[cnt] = point[0];
    }
};
Poly poly;
const Point zero(0, 0);
int N, W, H;
void cut(Point p[], int& cnt, const Point& a, const Point& b) {
    Point pp[1010];
    int n = 0;
    p[cnt] = p[0];
    for (int i = 0; i < cnt; i++) {
        if (sig((b - a).cross(p[i] - a)) > 0)
            pp[n++] = p[i];
        if (sig((b - a).cross(p[i] - a)) != sig((b - a).cross(p[i + 1]
- a)))
            pp[n++] = getSol(p[i], p[i + 1], a, b);
    }
    cnt = 0;
    for (int i = 0; i < n; i++) {
        if (!i || !(pp[i] == pp[i - 1]))
            p[cnt++] = pp[i];
    }
    while (cnt > 1 && p[cnt - 1] == p[0])
        cnt--;
}
double S(Point p[], int cnt) {
    p[cnt] = p[0];
    double s = 0;
    for (int i = 0; i < cnt; i++) {
        s += p[i].cross(p[i + 1]);
    }
    s /= 2;
    return s;
}
int sign(const Point& from, const Point& to) {
    Point v = to - from;
    double t = atan2(v.y, v.x);
    if (sig(t - PI / 2) == 0 || sig(t + PI / 2) == 0)
        return 0;
    if (sig(t + PI / 2) > 0 && sig(t - PI / 2) < 0)
        return -1;
    return 1;
}
double getS(Point t, Point s) {
    Point p[100];
    int cnt = 0;

    for (int i = 0; i < poly.cnt; i++)
        p[cnt++] = poly.point[i];
    cut(p, cnt, t, s);
    return fabs(S(p, cnt));
}
Point points[10010];
double result[110][110];
void init() {
    for (int i = 0; i < N; i++) {
        scanf("%lf%lf", &points[i].x, &points[i].y);
    }
    points[N] = points[0];
}
void work() {
    memset(result, 0, sizeof(result));
    for (int i = 0; i < N; i++) {
        Point from = points[i];
        Point to = points[i + 1];
        int s = sign(from, to);
        if (s == 0)
            continue;
        if (s == 1)
            swap(from, to);
        for (int j = from.x; j < to.x; j++) {
            for (int k = 0; k < H; k++) {
                poly.init();
                poly.addPoint(Point(j, k));
                poly.addPoint(Point(j + 1, k));
                poly.addPoint(Point(j + 1, k + 1));
                poly.addPoint(Point(j, k + 1));
                poly.finish();
                int flag = 0;
                for (int l = 0; l < 4; l++) {
                    if (sig((to - from).cross(poly.point[l])) > 0) {
                        flag = 1;
                        break;
                    }
                }
                result[j][k] += s * getS(from, to);
            }
        }
    }
    for (int j = H - 1; j >= 0; j--) {
        for (int i = 0; i < W; i++) {
            if (sig(result[i][j]) >= 0 && sig(result[i][j] - 0.25) <
0)
                putchar('.');
            if (sig(result[i][j] - 0.25) >= 0 && sig(result[i][j] -
0.5) < 0)
                putchar('+');
            if (sig(result[i][j] - 0.5) >= 0 && sig(result[i][j] -
0.75) < 0)
                putchar('o');
            if (sig(result[i][j] - 0.75) >= 0 && sig(result[i][j] - 1)
< 0)
                putchar('$');
            if (sig(result[i][j] - 1) == 0)
                putchar('#');
        }
        puts("");
    }
}
```

圆与多边形面积交

```
const Point zero(0, 0);
struct Poly {
    Point point[1010];
    int cnt;
    void init() {
        cnt = 0;
    }
    void addPoint(const Point& p) {
        point[cnt++] = p;
    }
    void finish() {
        point[cnt] = point[0];
    }
};
Poly poly;
struct Circle {
    Point center;
    double r;
    int intersec(const Point& p, const Point& q) {
        return sig(center.dist(p, q) - r);
    }
    void intersec(const Point& p, const Point& q, Point &r1, Point
&r2) {
        double d = center.dist(p, q);
        double dis = (zero - p).dot(q - p) / p.dist(q);
        Point D = trans(q - p, 0, dis / p.dist(q)) + p;
        Point t = trans(q - p, 0, sqrt(r * r - d * d) / p.dist(q));
        r1 = D + t;
        r2 = D - t;
    }
    int intLine(const Point& p, const Point& q) {
        double t1 = (q - p).dot(center - p);
        double t2 = (p - q).dot(center - q);
        return sig(center.dist(p, q) - r) < 0 && sig(t1) >= 0 &&
sig(t2) >= 0;
    }
};
Circle circle;
int isin(const Point& p) {
    return sig(circle.center.dist(p) - circle.r) <= 0;
}
int onSeg(const Point& p, const Point& a, const Point& b) {
    return sig((a - p).dot(b - p)) < 0;
}
double getS(const Point& from, const Point& to) {
    if (!sig(from.cross(to)))
        return 0.0;
    Point t1, t2;
    circle.intersec(from, to, t1, t2);
    Point pnt[10];
    int cnt = 0;
    pnt[cnt++] = from;
    if (onSeg(t1, from, to))
        pnt[cnt++] = t1;
    if (onSeg(t2, from, to))
        pnt[cnt++] = t2;
    pnt[cnt++] = to;
    if (cnt == 4 && sig((pnt[2] - pnt[1]).dot(pnt[1] - pnt[0])) < 0)
        swap(pnt[1], pnt[2]);
    double res = 0;
    for (int i = 0; i < cnt - 1; i++) {
        Point a = pnt[i];
        Point b = pnt[i + 1];
        double theta = fixacos(a.dot(b) / a.dist(zero) /
b.dist(zero));

        if (!isin(pnt[i]) || !isin(pnt[i + 1])) {
            res += circle.r * circle.r * theta;
        } else
            res += fabs(pnt[i].cross(pnt[i + 1]));
    }
    return res;
}
double interS() {
    double s = 0;
    for (int i = 0; i < poly.cnt; i++) {
        int sign = sig(poly.point[i].cross(poly.point[i + 1]));
        s += getS(poly.point[i], poly.point[i + 1]) * sign;
    }
    s = fabs(s);
    return s / 2;
}
int N;
void init() {
    circle.center = zero;
    double p, q, r, s;
    scanf("%d", &N);
    poly.init();
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf", &p, &q);
        poly.addPoint(Point(p, q));
    }
    poly.finish();
}
void work() {
    printf("%.2lf\n", interS());
}
int main() {
    int num = 0;
    while (scanf("%lf", &circle.r) != EOF)
    {
        init();
        work();
    }
    return 0;
}
```

两凸包最短距离

```
struct Point {
    double x, y;
    double angle(const Point& p) const {
        Point zero(0, 0);
        return this->dot(p) / this->dist(zero) / p.dist(zero);
    }
    bool operator<(const Point& p) const {
        const Point zero(0, 0);
        double t = this->cross(p);
        if (sig(t))
            return sig(t) > 0;
        return sig(this->dist(zero) - p.dist(zero)) > 0;
    }
    bool inRange(const Point& p, const Point& q) {
        double s = (*this - p).dot(q - p);
        double t = (*this - q).dot(p - q);
        return sig(s * t) >= 0;
    }
};

struct Poly {
    Point point[maxn];
    int cnt, R;
    Point result[maxn];
    Point temp[maxn];
    void init() {
        cnt = 0;
    }
    void addPoint(const Point& p) {
        point[cnt++] = p;
    }
    void reverse() {
        for (int i = 0; i < cnt / 2; i++) {
            Point t = point[i];
            point[i] = point[cnt - 1 - i];
            point[cnt - 1 - i] = t;
        }
    }
    void adjust() {
        point[cnt] = point[0];
        point[cnt + 1] = point[1];
        int isRight = 0;
        for (int i = 2; i < cnt; i++) {
            if ((point[i - 1] - point[i - 2]).cross(point[i] - point[i
- 1])
                    < 0) {
                isRight = 1;
                break;
            }
        }
        if (isRight)
            reverse();
        point[cnt] = point[0];
        point[cnt + 1] = point[1];
    }
    void buildConvexHull() {
        memcpy(temp + 1, point, sizeof(point));
        int N = cnt;
        for (int i = 1; i <= N; i++) {
            if (sig(temp[1].y - temp[i].y) > 0 || (sig(temp[1].y -
temp[i].y)
                    == 0 && sig(temp[1].x - temp[i].x) > 0))
                swap(temp[1], temp[i]);
        }
        Point zero(0, 0), source(temp[1]);
        for (int i = 2; i <= N; i++)
            temp[i] = temp[i] - temp[1];
        sort(temp + 2, temp + 1 + N);
        temp[1] = zero;
        int n = N;
        N = 1;
        for (int i = 2; i <= n;) {
            Point current = temp[i];
            temp[++N] = current;
            for (; i <= n && !sig(current.cross(temp[i])); i++)
                ;
        }
        R = 0;
        result[R++] = temp[1];
        result[R++] = temp[2];
        for (int i = 3; i <= N; i++) {
            while (R >= 2 && sig((result[R - 1] - result[R -
2]).cross(temp[i]
                    - result[R - 2])) < 0)
                R--;
            result[R++] = temp[i];
        }
        for (int i = 0; i < R; i++)
            result[i] = result[i] + source;
    }
};
struct Line {
    Point a, b;
    double dist(Point& p) {
        if (p.inRange(a, b))
            return fabs((p - a).cross(b - a) / a.dist(b));
        return min(p.dist(a), p.dist(b));
    }
    double dist(Line& p) {
        if (p.a.inRange(a, b))
            return this->dist(p.a);
        if (p.b.inRange(a, b))
            return this->dist(p.b);
        return min(min(p.a.dist(a), p.a.dist(b)), min(p.b.dist(a),
p.b.dist(b)));
    }
};
Poly P, Q;
int N, M;
void init() {
    double p, q;
    P.init();
    Q.init();
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf", &p, &q);
        Point tmp(p, q);
        P.addPoint(tmp);
    }
    for (int i = 1; i <= M; i++) {
        scanf("%lf%lf", &p, &q);
        Point tmp(p, q);
        Q.addPoint(tmp);
    }
    P.buildConvexHull();
    Q.buildConvexHull();
}
double degree(const Point& p, const Point& q, const Point& r, const
Point& s) {
    Point a(q - p);
    Point b(s - r);
    return a.angle(b);
}
void work() {
    int rp = 0, rq = 0;
    for (int i = 0; i < Q.R; i++) {
        if (sig(Q.result[rq].y - Q.result[i].y) < 0 ||
(sig(Q.result[rq].y
                - Q.result[i].y) == 0 && sig(Q.result[rq].x -
Q.result[i].x)
                < 0))
            rq = i;
    }
    Line cp;
    cp.a = P.result[rp];
    cp.b = cp.a;
    cp.b.x += 1;
    Line cq;
    cq.a = Q.result[rq];
    cq.b = cq.a;
    cq.b.x -= 1;
    double res = P.result[rp].dist(Q.result[rq]);
    int currentP = rp, currentQ = rq;
    do {
        int np = (currentP + 1) % P.R;
        int nq = (currentQ + 1) % Q.R;
        double cosP = degree(cp.a, cp.b, P.result[currentP],
P.result[np]);
        double cosQ = degree(cq.a, cq.b, Q.result[currentQ],
Q.result[nq]);
        if (sig(cosP - cosQ) > 0) {
            cp.a = P.result[currentP];
            cp.b = P.result[np];
            cq.b = cq.a + cp.a - cp.b;
            currentP = np;
            res = min(res, cp.dist(Q.result[currentQ]));
        } else if (sig(cosP - cosQ) < 0) {
            cq.a = Q.result[currentQ];
            cq.b = Q.result[nq];
            cp.b = cp.a + cq.a - cq.b;
            currentQ = nq;
            res = min(res, cq.dist(P.result[currentP]));
        } else {
            cp.a = P.result[currentP];
            cp.b = P.result[np];
            cq.a = Q.result[currentQ];
            cq.b = Q.result[nq];
            currentP = np;
            currentQ = nq;
            res = min(res, cp.dist(cq));
        }
    } while (currentP != rp || currentQ != rq);
    printf("%.5lf\n", res);
}
```

点集最大面积三角形

```
struct Point {
    int x, y;
    inline bool operator<(const Point& p) const {
        const Point zero(0, 0);
        int t = this->cross(p);
        if (t)
            return t > 0;
        return this->dist(zero) - p.dist(zero) > 0;
    }
};
Point point[maxn], stack[maxn];
int N, top;
inline int S(const Point& p, const Point& q, const Point& r) {
    return abs(p.cross(q) + q.cross(r) + r.cross(p));
}
int num[maxn * 2];
void work() {
    convexHull();
    for (int i = 0; i < top * 2; i++)
        num[i] = i % N;
    int ans = 0;
    for (int i = 0; i < top; i++) {
        int j = num[i + 1];
        int k = num[j + 1];
        while (k != i && S(stack[i], stack[j], stack[num[k + 1]]) >
S(stack[i],
                stack[j], stack[k]))
            k = num[k + 1];
        int kk = num[k + 1];
        while (j != kk && k != i) {
            ans = max(ans, S(stack[i], stack[j], stack[k]));
            while (k != i && S(stack[i], stack[j], stack[num[k + 1]])
> S(
                    stack[i], stack[j], stack[k]))
                k = num[k + 1];
            j = num[j + 1];
        }
    }
    printf("%.2lf\n", ans / 2.0);
}
```

最近圆对

```
struct Circle {
    Point center;
    double r;
    int id;
    bool operator<(const Circle& p) const {
        if (sig(center.y - p.center.y))
            return sig(center.y - p.center.y) < 0;
        if (sig(center.x - p.center.x))
            return sig(center.x - p.center.x) < 0;
        return id < p.id;
    }
    bool operator==(const Circle& p) const {
        return sig(center.y - p.center.y) == 0 && sig(center.x -
p.center.x)
                == 0 && id == p.id;
    }
};
Circle circle[100010];
int l[100010], r[100010], N;
bool compL(int p, int q) {
    double t = circle[p].center.x - circle[p].r;
    double s = circle[q].center.x - circle[q].r;
    return sig(t - s) < 0;
}
bool compR(int p, int q) {
    double t = circle[p].center.x + circle[p].r;
    double s = circle[q].center.x + circle[q].r;
    return sig(t - s) < 0;
}
void init() {
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf%lf", &circle[i].center.x, &circle[i].center.y,
                &circle[i].r);
        l[i] = i;
        r[i] = i;
        circle[i].id = i;
    }
    sort(l + 1, l + 1 + N, compL);
    sort(r + 1, r + 1 + N, compR);
}
int intersect(set<Circle>& S, const Circle& p) {
    if (S.size() <= 1)
        return 0;
    set<Circle>::iterator ite = S.find(p);
    set<Circle>::iterator tmp = ite;
    if (ite != S.begin()) {
        tmp--;
        if (sig(tmp->center.dist(p.center) - tmp->r - p.r) < 0)
            return 1;
    }
    tmp = ite;
    tmp++;
    if (tmp != S.end()) {
        if (sig(tmp->center.dist(p.center) - tmp->r - p.r) < 0)
            return 1;
    }
    return 0;
}
int result[100010];
int check() {
    memset(result, 0, sizeof(result));
    set<Circle> S;
    int i = 1, j = 1;
    while (i <= N && j <= N) {
        if (i == N + 1) {
            if (S.find(circle[r[j]]) != S.end())
                S.erase(circle[r[j]]);
            j++;
        } else if (j == N + 1) {
            if (S.find(circle[l[i]]) != S.end()) {
                result[l[i]] = 1;
                i++;
                continue;
            }
            S.insert(circle[l[i]]);
            if (intersect(S, circle[l[i]])) {
                return 0;
            }
            i++;
        } else if (i <= N && sig(circle[l[i]].center.x -
circle[l[i]].r
                - (circle[r[j]].center.x + circle[r[j]].r)) < 0) {
            if (S.find(circle[l[i]]) != S.end()) {
                result[l[i]] = 1;
                i++;
                continue;
            }
            S.insert(circle[l[i]]);
            if (intersect(S, circle[l[i]])) {
                return 0;         }
            i++;
        } else {
            if (S.find(circle[r[j]]) != S.end())
                S.erase(circle[r[j]]);
            j++;
        }
    }
    return 1;
}
void work() {
    double low = 0, high = 1e8, ans = -1;
    while (sig(high - low) > 0) {
        double mid = (low + high) / 2;
        for (int i = 1; i <= N; i++)
            circle[i].r += mid / 2;
        if (check()) {
            ans = mid;
            low = mid;
        } else
            high = mid;
        for (int i = 1; i <= N; i++)
            circle[i].r -= mid / 2;
    }
    printf("%.6lf\n", ans);
}
```

不被嵌套圆

```
struct Circle {
    Point center;
    double r;
    int id;
    bool operator<(const Circle& p) const {
        if (sig(center.y - p.center.y))
            return sig(center.y - p.center.y) < 0;
        if (sig(center.x - p.center.x))
```

```
            return sig(center.x - p.center.x) < 0;
        return id < p.id;
    }
    bool operator==(const Circle& p) const {
        return sig(center.y - p.center.y) == 0 && sig(center.x -
p.center.x)
                == 0;
    }
};

Circle circle[40010];
int l[40010], r[40010], N;
bool compL(int p, int q) {
    double t = circle[p].center.x - circle[p].r;
    double s = circle[q].center.x - circle[q].r;
    return sig(t - s) < 0;
}
bool compR(int p, int q) {
    double t = circle[p].center.x + circle[p].r;
    double s = circle[q].center.x + circle[q].r;
    return sig(t - s) < 0;
}
void init() {
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf%lf", &circle[i].r, &circle[i].center.x,
                &circle[i].center.y);
        l[i] = i;
        r[i] = i;
        circle[i].id = i;
    }
    sort(l + 1, l + 1 + N, compL);
    sort(r + 1, r + 1 + N, compR);
}
int intersect(set<Circle>& S, const Circle& p) {
    if (S.size() <= 1)
        return 0;
    set<Circle>::iterator ite = S.find(p);
    set<Circle>::iterator tmp = ite;
    if (ite != S.begin()) {
        tmp--;
        if (sig(tmp->center.dist(p.center) - tmp->r - p.r) < 0)
            return 1;
    }
    tmp = ite;
    tmp++;
    if (tmp != S.end()) {
        if (sig(tmp->center.dist(p.center) - tmp->r - p.r) < 0)
            return 1;
    }
    return 0;
}
int result[40010];
void work() {
    memset(result, 0, sizeof(result));
    set<Circle> S;
    int i = 1, j = 1;
    while (i <= N && j <= N) {
        if (i == N + 1) {
            if (S.find(circle[r[j]]) != S.end())
                S.erase(circle[r[j]]);
            j++;
        } else if (j == N + 1) {
            if (S.find(circle[l[i]]) != S.end()) {
                result[l[i]] = 1;
                i++;
                continue;
            }
            S.insert(circle[l[i]]);
            if (intersect(S, circle[l[i]])) {
                S.erase(circle[l[i]]);
                result[l[i]] = 1;
            }
            i++;
        } else if (i <= N && sig(circle[l[i]].center.x - 
circle[l[i]].r
                - (circle[r[j]].center.x + circle[r[j]].r)) < 0) {
            if (S.find(circle[l[i]]) != S.end()) {
                result[l[i]] = 1;
                i++;
                continue;
            }
            S.insert(circle[l[i]]);
            if (intersect(S, circle[l[i]])) {
                S.erase(circle[l[i]]);
                result[l[i]] = 1;
            }
            i++;
        } else {
            if (S.find(circle[r[j]]) != S.end())
                S.erase(circle[r[j]]);
            j++;
        }
    }
    int cnt = 0;
    for (int i = 1; i <= N; i++)
        cnt += result[i];
    cnt = N - cnt;
    printf("%d\n", cnt);
    for (int i = 1, j = 1; i <= N; i++) {
        if (!result[i]) {
            printf("%d", i);
            if (j == cnt)
                putchar('\n');
            else
                putchar(' ');
            j++;
        }
    }
}
```

最小包围圆

```
struct Circle {
    Point center;
    double r;
    Circle() {}
    Circle(const Point& p, const Point& q, const Point& r) {
        Point t = p + q;
        t.x /= 2;
        t.y /= 2;
        Point v = trans((p - t), PI / 2, 1) + t;
        Point u = q + r;
        u.x /= 2;
        u.y /= 2;
        Point w = trans((q - u), PI / 2, 1) + u;
        center = getSol(t, v, u, w);
        this->r = center.dist(p);
    }
    Circle(const Point& p, const Point& q) {
        center = p + q;
        center.x /= 2;
        center.y /= 2;
        r = center.dist(p);
    }
    int isin(const Point& p) const {
        return sig(p.dist(center) - r) <= 0;
    }
};
Point point[110];
int N;
void init() {
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf", &point[i].x, &point[i].y);
    }
    for (int i = N; i >= 2; i--) {
        int k = rand() % i + 1;
        swap(point[i], point[k]);
    }
}
void work() {
    Circle currentCircle(point[1], point[2]);
    for (int i = 3; i <= N; i++) {
        if (!currentCircle.isin(point[i])) {
            Circle circle1(point[1], point[i]);
            for (int j = 2; j <= i - 1; j++) {
                if (!circle1.isin(point[j])) {
                    Circle circle2(point[j], point[i]);
                    for (int k = 1; k <= j - 1; k++) {
                        if (!circle2.isin(point[k])) {
                            Circle tmp(point[j], point[i], point[k]);
                            circle2 = tmp;
                        }
                    }
                    circle1 = circle2;
                }
            }
            currentCircle = circle1;
        }
    }
    printf("%.2lf %.2lf %.2lf\n", currentCircle.center.x,
            currentCircle.center.y, currentCircle.r);
}
```

最近点对

```
using namespace std;
const int Max = 200001;
const double inf = 1e-8;
const double off = 1e100;
struct Point {
    double x, y;
};
Point p[Max];
inline int dbcmp(double tp) {
    return tp < -inf ? -1 : tp > inf;
}
inline double Distance(Point a, Point b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y -
b.y));
}
bool cmpx(Point a, Point b) {
    return a.x < b.x;
}
bool cmpy(int a, int b) {
    return p[a].y < p[b].y;
}
int ypos[Max];
double closedis(Point *p, int l, int r) {
    if (l == r)
        return off;
    if (l + 1 == r)
        return Distance(p[l], p[r]);
    int mid = (l + r) / 2;
    double ans = min(closedis(p, l, mid), closedis(p, mid + 1, r));
    int num = 0;
    for (int i = l; i <= r; i++) {
        if (dbcmp(fabs(p[i].x - p[mid].x) - ans) <= 0)
            ypos[num++] = i;
    }
    sort(ypos, ypos + num, cmpy);
    for (int i = 0; i < num; i++) {
        int k = 0;
        for (int j = i + 1; j < num; j++) {
            if (dbcmp(fabs(p[ypos[i]].y - p[ypos[j]].y) - ans) >= 0)
                break;
            else {
                ans = min(ans, Distance(p[ypos[i]], p[ypos[j]]));
                k++;
            }
            if (k > 6)
                break;
        }
    }
    return ans;
}

int main() {
    int n;
    while (scanf("%d", &n), n) {
        for (int i = 0; i < n; i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        sort(p, p + n, cmpx);
        //printf("%.2lf\n", closedis(p, 0, n - 1) / 2.0);

    }
}
```

线段树求举行面积并

```
const int maxn = 10010;
double v[maxn];
struct Seg {
    int x, y, z;
    int value;
    bool operator<(const Seg& p) const {
        return x < p.x;
    }
};
Seg seg[410];
struct Rect {
    double x1, y1, x2, y2;
};
```

```
Rect rect[410];
template<class T>
struct SegNode {
    T key;
    int flag;
    int left, right;
    double sum;
    T add;

    int mid() {
        return (left + right) >> 1;
    }
    void update() {
        if (key)
            sum = v[right + 1] - v[left];
    }
};
template<class T>
struct SegTree {
    SegNode<T> tree[5 * maxn];
    void init(int left, int right, int idx, T value[]) {
        tree[idx].left = left;
        tree[idx].right = right;
        tree[idx].flag = 0;
        tree[idx].add = 0;
        tree[idx].sum = 0;
        if (left == right) {
            tree[idx].key = value[left];
            return;
        }
        int mid = tree[idx].mid();
        init(left, mid, idx << 1, value);
        init(mid + 1, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    void update(int left, int right, int idx, T value) {
        //It's a sub-interval, update it here.
        if (left <= tree[idx].left && right >= tree[idx].right) {
            tree[idx].key += value;
            tree[idx].update();
            tree[idx].flag = 1;
            push_up(idx);
            return;
        }
        push_down(idx);
        int mid = tree[idx].mid();
        if (left <= mid)
            update(left, right, idx << 1, value);
        if (mid < right)
            update(left, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    double query(int left, int right, int idx) {
        //Query result here.
        if (left == tree[idx].left && right == tree[idx].right) {
            return tree[idx].sum;
        }
        push_down(idx);
        int mid = tree[idx].mid();
        if (right <= mid)
            return query(left, right, idx << 1);
        else if (left > mid)
            return query(left, right, (idx << 1) + 1);
        else {
            return (query(left, mid, idx << 1) + query(mid + 1, right,
(idx
                    << 1) + 1));
        }
    }
    void push_down(int idx) {}
    void push_up(int idx) {
        if (tree[idx].key)
            tree[idx].update();
        else
            tree[idx].sum = tree[idx << 1].sum + tree[(idx << 1) +
1].sum;
    }
};
int N, C, S;
SegTree<int> tree;
int BS(double t) {
    int low = 1, high = C;
    while (low <= high) {
        int mid = (low + high) >> 1;
        if (v[mid] == t)
            return mid;
        else if (v[mid] < t)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
int tmp[10010];
void init() {
    C = 0;
    S = 0;
    for (int i = 1; i <= N; i++) {
        scanf("%lf%lf%lf%lf", &rect[i].x1, &rect[i].y1, &rect[i].x2,
                &rect[i].y2);
        v[++C] = rect[i].x1;
        v[++C] = rect[i].y1;
        v[++C] = rect[i].x2;
        v[++C] = rect[i].y2;
    }
    sort(v + 1, v + 1 + C);
    int c = C;
    C = 0;
    for (int i = 1; i <= c;) {
        double current = v[i];
        v[++C] = current;
        for (; i <= c && current == v[i]; i++)
            ;
    }
    for (int i = 1; i <= N; i++) {
        ++S;
        int p = BS(rect[i].x1), q = BS(rect[i].y1), r =
BS(rect[i].x2), s = BS(
                rect[i].y2);
        seg[S].x = p;
        seg[S].y = q;
        seg[S].z = s;
        seg[S].value = 1;
        ++S;
        seg[S].x = r;
        seg[S].y = q;
        seg[S].z = s;
        seg[S].value = -1;
    }
    sort(seg + 1, seg + 1 + S);
    tree.init(1, C, 1, tmp);
}
void work(int num) {
    double res = 0;
    int last = seg[1].x;
    for (int i = 1; i <= S;) {
        res += (v[seg[i].x] - v[last]) * tree.query(1, C, 1);
        last = seg[i].x;
        int current = seg[i].x;
        for (; i <= S && current == seg[i].x; i++) {
            tree.update(seg[i].y, seg[i].z - 1, 1, seg[i].value);
        }
    }
    printf("Test case #%d\n", num);
    printf("Total explored area: %.2lf\n", res);
}
```