几何基础

```
int sign(double a)
{
    if(abs(a) < eps)   return 0;
    return a > 0 ? 1 : -1;
}

int cmp(double a, double b)
{
    if(abs(a - b) < eps)   return 0;
    return a > b ? 1 : -1;
}

PDD operator + (PDD a, PDD b)
{
    return { a.x + b.x, a.y + b.y };
}

PDD operator - (PDD a, PDD b)
{
    return { a.x - b.x, a.y - b.y };
}

PDD operator * (PDD a, double b)
{
    return { a.x * b, a.y * b };
}

double operator * (PDD a, PDD b)
```

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

double operator & (PDD a, PDD b)
{
    return a.x * b.x + a.y * b.y;
}

PDD operator / (PDD a, double b)
{
    return { a.x / b, a.y / b };
}

double get_len(PDD a)
{
    return sqrt(a.x * a.x + a.y * a.y);
}

double get_dist(PDD a, PDD b)
{
    double dx = a.x - b.x, dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}

double area(PDD a, PDD b, PDD c)
{
    return (b - a) * (c - a);
}
```

向量夹角

```
double get_angle(PDD a, PDD b)              //两向量夹角
{
    return acos((a & b) / get_len(a) / get_len(b));
}
```

顺时针旋转

```
PDD rotate(PDD a, double b)
{
    return {a.x * cos(b) + a.y * sin(b), -a.x * sin(b) + a.y * cos(b)};
}
```

两直线交点(sign(v * w) != 0)

```
PDD get_line_intersection(PDD p, PDD v, PDD q, PDD w)                    //两直线交点(点向式)
{
    PDD u = p - q;
    double t = w * u / (v * w);
    return p + v * t;
}
```

点到直线距离

```
double distance_to_line(PDD p, PDD a, PDD b)                    //点 p 到直线(两点式)的距离
{
    PDD v1 = b - a, v2 = p - a;
    return abs(v1 * v2 / get_len(v1));
}
```

点到线段距离

```
double distance_to_segment(PDD p, PDD a, PDD b)                 //点 p 到线段的距离
{
    if (a == b) return get_len(p - a);

    PDD v1 = b - a, v2 = p - a, v3 = p - b;
    if (sign(v1 & v2) < 0) return get_len(v2);
    if (sign(v1 & v3) > 0) return get_len(v3);
    return distance_to_line(p, a, b);
}
```

点在直线上的投影

```
PDD get_line_projection(PDD p, PDD a, PDD b)                    //点 p 到直线(两点式)上的投影点
{
    PDD v1 = b - a, v2 = p - a;
    return a + v1 * ((v1 & v2) / (v1 & v1));
}
```

点是否在线段上

```
bool on_segment(PDD p, PDD a, PDD b)                    //点 p 是否在线段上
{
    return !sign((p - a) * (p - b)) && sign((p - a) & (p - b)) <= 0;
}
```

线段是否相交(不包括端点)

```
bool segment_intersection(PDD a1, PDD a2, PDD b1, PDD b2)       //线段与线段是否相交
{
    double c1 = (a2 - a1) * (b1 - a1), c2 = (a2 - a1) * (b2 - a1);
    double c3 = (b2 - b1) * (a1 - b1), c4 = (b2 - b1) * (a2 - b1);
    return sign(c1) * sign(c2) < 0 && sign(c3) * sign(c4) < 0;
}
```

射线是否相交

```
bool ray_intersection(PDD a1, PDD a2, PDD b1, PDD b2)   //射线与射线是否相交
{
    int s = sign((a2 - a1) * (b2 - b1));
    if(!s)  return false;
    return sign(area(b1, b2, a1)) == s && sign(area(a1, a2, b1)) == -s;
}
```

多边形面积(任意多边形)

```
double area(PII* p, int n)
{
    double s = 0;
    for (int i = 1; i <= n; i ++ )
         s += area(p[1], p[i], p[i % n + 1]);
    return s / 2;
}
```

点与多边形的关系

```
int get_point_polygon(PDD* p, int n, PDD P)     //逆时针或顺时针存储
{
    int res = 1;
    for(int i = 0; i < n; i ++)
    {
        PDD p1 = p[i], p2 = p[(i + 1) % n];

        if(on_segment(P, p1, p2))               //点在多边形线段上
            return 0;

        if(p1.y > p2.y && area(p1, p2, P) < 0  || p1.y < p2.y && area(p1, p2, P)
> 0)
            if(sign(p1.y - P.y) > 0 != sign(p2.y - P.y) > 0)    //多边形顶点上移
                res *= -1;     //min(p1.y, p2.y) < P.y <= max(p1.y, p2.y)
    }
    return res;                 //1：外面，  -1：内部
}
```

点与凸多边形的关系

```
string get_point_convexpolygon(PII* p, int n, PII P)      //逆时针存储
```

```
{
    p[n + 1] = p[1];

    if(area(p[1], p[2], P) < 0 || area(p[1], p[n], P) > 0)
        return "OUT";
    else
    {
        int l = 1, r = n;
            while(l < r)
            {
                int mid = l + r + 1 >> 1;
                if(area(p[1], p[mid], P) >= 0)    l = mid;
                else    r = mid - 1;
            }
            if(on_segment(P, p[l - 1], p[l]) || on_segment(P, p[l], p[l + 1]))
               return "ON";
            else if(area(p[l], p[l + 1], P) > 0)
                return "IN";
            else
                return "OUT";
    }
}
```

直线与凸多边形的关系

```
bool line_intersect_convexpolygon(Line& line, PII* p, int n)    //p：逆时针存储
{
    for(int i = 1; i <= n; i ++)
    {
        PII l = p[i % n + 1] - p[i];
        angle[i] = atan2(l.y, l.x);

        if(angle[i] < -PI / 2)
            angle[i] += 2 * PI;
    }

    PII l1 = line.st - line.ed, l2 = line.ed - line.st;
    double angle1 = atan2(l1.y, l1.x), angle2 = atan2(l2.y, l2.x);
    if(angle1 < -PI / 2)
        angle1 += 2 * PI;
    if(angle2 < -PI / 2)
        angle2 += 2 * PI;

    //平行与 line 且与凸多边形相切的两条直线
    int a = lower_bound(angle + 1, angle + 1 + n, angle1) - angle,
        b = lower_bound(angle + 1, angle + 1 + n, angle2) - angle;

    // line 是否在两条切线中间
    return sign((line.ed - line.st) * (p[a] - line.st)) * sign((line.ed -
line.st) *(p[b] - line.st)) <= 0;
}
```

过点求凸多边形的两个切点

```
int get(int l, int r, PII P, int w)         //求切点
{
    while(l < r)
    {
        int mid = l + r >> 1;           //top为凸多边形点的数量
        if(area(P, p[mid], p[mid % top + 1]) * w > 0)     r = mid;
        else l = mid + 1;
    }
    return l;
}

int bord(int l, int r, PII P, int w)    //找分割点
{
    while(l < r)
    {
        int mid = l + r >> 1;
        if((p[mid].x - P.x) * w > 0)    r = mid;
        else l = mid + 1;
    }
    return l;
}

PII get_polygon_tangent(PII* p, int n, PII P)      //凸多边形p
{
    p[n + 1] = p[1];

    if(P.x < p[1].x)     //左侧
        int L = get(1, ttop, P, 1), R = get(ttop, n + 1, P, -1);
    else if(P > p[ttop].x)     //右侧
        int L = get(1, ttop, P, -1), R = get(ttop, n + 1, P, 1);
    else if(area(p[1], p[ttop], P) > 0)      //上侧
    {
        int Mid = bord(ttop, n + 1, P, -1);
        int L = Mid > ttop ? get(ttop, Mid - 1, P, -1) : Mid, R = get(Mid, n +
1, P, 1);
    }
    else                //下侧
    {
        int Mid = bord(s, 1, ttop, P, 1);
        int L = Mid > 1 ? get(1, Mid - 1, P, -1) : 1, R = get(Mid, ttop, P, 1);
    }
}
```

圆与直线交点

```
//圆与直线(两点式)的交点
int circle_intersect_line(Circle& a, PDD& b, PDD& c, PDD& p1, PDD& p2)
{
    double d = distance_to_line(a.O, b, c);

    if(cmp(d, a.r) > 0)           //相离
        return 0;

    PDD e = get_line_intersection(a, b - a, a.O, rotate(b - a, PI / 2));
```

```
    double len = sqrt(a.r * a.r - d * d);
    p1 = e + norm(a - b) * len, p2 = e + norm(b - a) * len;

    if(!cmp(d, a.r))              //相切
        return 1;
    return 2;                //相交
}
```

两圆交点

```
int circle_intersect_circle(Circle& a, Circle& b, PDD& p1, PDD& p2)
{
    double d = get_dist(a.O, b.O);
    if(cmp(d, a.r + b.r) > 0 || cmp(d, abs(a.r - b.r)) < 0)     //两圆外离,内含
        return 0;

    PDD v = b.O - a.O;
    double A = acos((a.r * a.r + d * d - b.r * b.r) / (2 * d * a.r)),
        B = atan2(v.y, v.x);

    ang1 = A + B, ang2 = -A + B;

    p1 = a.O + (PDD) { a.r* cos(ang1), a.r* sin(ang1) },
    p2 = a.O + (PDD) { a.r* cos(ang2), a.r* sin(ang2) };

    if(!cmp(d, a.r + b.r) || !cmp(d, abs(a.r - b.r)))       //外切与内切
        return 1;
    return 2;
}
```

n个圆的面积交

```
int get_circle_circle_point(Circle& a, Circle& b, double& angle1, double&
angle2)
{
    double d = get_dist(a.O, b.O);
    if(cmp(d, a.r + b.r) > 0 || cmp(d, abs(a.r - b.r)) < 0)     //两圆外离,内含
        return 0;

    PDD v = b.O - a.O;
    double angle = acos((a.r * a.r + d * d - b.r * b.r) / (2 * d * a.r)),
        delta = atan2(v.y, v.x);

    angle1 = delta - angle, angle2 = delta + angle;

    if(angle1 < -PI)
        angle1 += 2 * PI;
    if(angle > PI)
        angle1 -= 2 * PI;
    if(angle2 < -PI)
        angle2 += 2 * PI;
    if(angle2 > PI)
        angle2 -= 2 * PI;
```

```
    if(!cmp(d, a.r + b.r) || !cmp(d, abs(a.r - b.r)))
        return 1;
    return 2;
}

double calc(Circle& c, double l, double r)     //积分
{
    double res = c.r * c.r * (r - l) + c.o.x * c.r * (sin(r) - sin(l)) -
        c.o.y * c.r * (cos(r) - cos(l));
    return res / 2;
}

bool compare(Data& a, Data& b)
{
    if(cmp(a.angle, b.angle))   return cmp(a.angle, b.angle) < 0;
    return a.state > b.state;
}

void Circle_Union(Circle &c, int n)     //area[i]:至少被覆盖i次的面积
{
    for(int i = 1; i <= n; i ++)
        for(int j = 1; j <= n; j ++)
            if(i != j && cmp(c[j].r, c[i].r) >= 0
                && cmp(get_dist(c[i].o, c[j].o), c[j].r - c[i].r) <= 0)
                c[i].state ++;

    for(int i = 1; i <= n; i ++)
    {
        int cnt = 0, t = 0;
        double angle1, angle2;
        for(int j = 1; j <= n; j ++)
            if(i != j && get_circle_circle_point(c[i], c[j], angle1, angle2) ==
2)
            {
                data_[cnt ++] = { angle1, 1 }, data_[cnt ++] = { angle2, -1 };
                if(cmp(angle1, angle2) > 0)
                    t ++;

            }
        data_[cnt ++] = { -PI, t }, data_[cnt ++] = { PI, -t };
        sort(data_, data_ + cnt, compare);

        int s = c[i].state + data_[0].state;
        for(int j = 1; j < cnt; j ++)
        {
            ans[s] += calc(c[i], data_[j - 1].angle, data_[j].angle);
            s += data_[j].state;
        }
    }
}
```