计算几何

```
const double eps = 10e-6;
const double pi = acos(-1);
int dcmp(double x) {
    if (fabs(x) < eps)  return 0;
    return x > 0 ? 1 : -1;
}

//计算几何基本内容 from LiGuanlin1124

struct vec{
    vec() {}
    vec(double x, double y) :x(x), y(y) {}
    double x, y;

    vec operator + (const vec& a)const      { return vec(x + a.x, y + a.y); }
    vec operator - (const vec& a)const      { return vec(x - a.x, y - a.y); }
    vec operator * (const double& a)const   { return vec(x*a, y*a); }
    vec operator / (const double& a)const   { return vec(x/a, y/a); }
    double operator * (const vec& a)const   { return x*a.x + y*a.y; }//点乘
    double operator ^ (const vec& a)const   { return x*a.y - y*a.x; }//叉乘

    bool operator < (const vec& a)const     { return x != a.x ? x < a.x : y < a.y;
}//右>上
    bool operator == (const vec & a)const   { return (!dcmp(a.x - x)) &&
(!dcmp(a.y - y)); }
};

double len(const vec& a) { return sqrt(a * a); }
double ang(const vec& a) { return atan2(a.y, a.x); }//与x轴夹角
double ang(const vec& a, const vec& b) { return acos(a * b / len(a) / len(b)); }

double triangle_s_sign(const vec a, const vec b, const vec c) { return (b - a) ^
(c - a) / 2; }
double triangle_s_unsign(const vec a, const vec b, const vec c) { return fabs((b -
a) ^ (c - a)) / 2; }

vec rotate(const vec a, double d) { return vec(a.x*cos(d) + a.y*sin(d), a.y*cos(d)
+ a.x*sin(d)); }
vec nol(const vec a) { return vec(-a.y, a.x) / len(a); }


struct line {
    line() {}
    line(vec x, vec v) :x(x), v(v) {}
    vec x;//点
    vec v;//方向向量
};
```

```
//相交
vec intersec(line a, line b)
{
    double t = (b.v ^ (b.x - a.x) / (b.v ^ a.v));
    return a.x + a.v * t;
}
//垂足
vec pedal(vec p, line l) { return l.x + l.v * ((p - l.x) * l.v) / (l.v * l.v); }

//点到直线
double dis_PL(vec p, line l) { return fabs(l.v ^ (p - l.x) / len(l.v)); }
//点到线段
double dis_PS(vec p, vec a1, vec a2)
{
    if (a1 == a2)   return len(p - a1);
    //垂足不在线段上
    vec v1 = a2 - a1, v2 = p - a1, v3 = p - a2;

    if (v1 * v2 <= 0)       return len(v2);
    else if (v1 * v3 >= 0)  return len(v3);
    else                    return (v1 ^ v2) / len(v1);
}
//是否相交(跨立)
bool cross(vec a1, vec a2, vec b1, vec b2)
{
    double d1 = (a2 - a1) ^ (b1 - a1);
    double d2 = (a2 - a1) ^ (b2 - a1);
    double d3 = (b2 - b1) ^ (a1 - b1);
    double d4 = (b2 - b1) ^ (a2 - b1);
    //叉乘异号在两侧
    return ((dcmp(d1) * dcmp(d2)) < 0) && ((dcmp(d3) * dcmp(d4)) < 0);
}
//是否共线
bool colline(vec a1, vec a2, vec b1, vec b2)
{
    vec v1 = a1 - a2;
    vec v2 = b1 - b2;
    return !dcmp(nol(v1) * v2) && (!dcmp(dis_PS(a1, b1, b2)) || !dcmp(dis_PS(a2,
b1, b2)));
}
//是否共端点
bool same_end(vec a1, vec a2, vec b1, vec b2)
{
    return (a1 == b1) || (a1 == b2) || (a2 == b1) || (a2 == b2);
}
//点在线上
bool on_line(vec p, line l) { return !dcmp(l.v^(l.x-p)); }
//点在线段上
bool on_seg(vec p, vec a1, vec a2)
{
    vec x = a1 - p;
    vec y = a2 - p;
    //点乘判断
    return dcmp(x * y) == 0 && dcmp(x ^ y) < 0;
```

```
}

//多边形
typedef vector<vec> polygon;

//从点集建凸包
polygon build_convex(int n, vec p[])
{
    vec* ans = new vec[n+1];
    sort(p, p + n);
    int count = 0;
    //下凸包
    for (int i = 0; i < n; i++)
    {
        while (count > 1 && ((ans[count - 1] - ans[count - 2]) ^ (p[i] - ans[count
- 2])) <= 0)    count--;
        ans[count++] = p[i];
    }
    //上凸包
    int lim = count;
    for (int i = n - 2; i >= 0; i--)
    {
        while (count > lim && ((ans[count - 1] - ans[count - 2]) ^ (p[i] -
ans[count - 2])) <= 0)  count--;
        ans[count++] = p[i];
    }
    if (count != 1) count--;

    polygon pol;
    for (int i = 0; i < count; i++)
        pol.push_back(ans[i]);
    free(ans);
    return pol;
}

//旋转卡壳
double rotate_stuck(polygon p)
{
    int q = 1;
    int n = p.size();
    p.push_back(p[0]);
    double ans = 0;
    for (int i = 0; i < n; i++)
    {
        //printf("%lf %lf\n", p[i].x, p[i].y);
        vec edg = p[i] - p[i + 1];
        while (fabs(edg ^ (p[i] - p[q+1])) > fabs(edg ^ (p[i] - p[q]))) q = (q +
1) % n;
        ans = max(ans, max(len(p[i]-p[q]),len(p[i+1]-p[q+1])));
    }
    return ans;
}
//多边形面积
double poly_s(polygen p)
```

```
{
    int q = 1; vec x = vec(0,0);
    int n = p.size();
    p.push_back(p[0]);
    double s = 0;
    for(int i=0;i<n;i++)
    {
        s+=(p[i]-x)^(p[i+1]-x);
    }
    return fabs(s);
}


//
struct intvec {
    llint x, y;
    intvec() {}
    intvec(llint x, llint y) :x(x), y(y) {}

    llint operator ^ (const intvec& a)const { return x * a.y - y * a.x; }
    intvec operator - (const intvec& a)const { return intvec(x - a.x, y - a.y); }
};
```