计算几何

ZW

```
struct Point {
    double x, y;
    Point(double x = 0, double y = 0):x(x), y(y){}
};

typedef Point Vector;

Vector operator + (const Vector& A, const Vector& B) { return Vector(A.x + B.x, A.y + B.y); }
Vector operator - (const Vector& A, const Vector& B) { return Vector(A.x - B.x, A.y - B.y); }
Vector operator * (const Vector& A, double p) { return Vector(A.x * p, A.y * p); }
Vector operator / (const Vector& A, double p) { return Vector(A.x / p, A.y / p); }
double operator ^ (const Vector& A, const Vector& B) {return A.x * B.y - A.y * B.x;}
bool operator < (const Point& a, const Point& b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

int dcmp(double x) {
    if(abs(x) < eps) return 0;
    return x < 0 ? -1 : 1;
}

bool operator == (const Point& a, const Point& b) {
    return !dcmp(a.x - b.x) && !dcmp(a.y - b.y);
}

double Dot(const Vector& A, const Vector& B) { return A.x * B.x + A.y * B.y; }
double Cross(const Vector& A, const Vector& B) { return A.x * B.y - A.y * B.x; }
double Length(const Vector& A) { return sqrt(Dot(A, A)); }
double Angle(const Vector& A, const Vector& B) { return acos(Dot(A, B) / Length(A) / Length(B)); }
double Area2(const Point& A, const Point& B, const Point& C) { return Cross(B - A, C - A); }

Vector Rotate(const Vector& A, double rad) {
    return Vector(A.x + cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
}

Vector Normal(const Vector& A) {
    double L = Length(A);
    return Vector(-A.y / L, A.x / L);
}

Point GetLineIntersection(const Point& P, const Vector& v, const Point& Q, const Vector& w) {
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);
    return P + v * t;
}

double DistanceToLine(const Point& P, const Point& A, const Point& B) {
    Vector v1 = B - A, v2 = P - A;
    return fabs(Cross(v1, v2) / Length(v1));
}

double DistanceToSegment(const Point& P, const Point& A, const Point& B) {
    if(A == B) return Length(P - A);
    Vector v1 = B - A, v2 = P - A, v3 = P - B;
    if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
    if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
    return fabs(Cross(v1, v2) / Length(v1));
}

Point GetLineProjection(const Point& P, const Point& A, const Point& B) {
    Vector v = B - A;
    return A + v * (Dot(v, P - A) / Dot(v, v));
}

bool SegmentProperIntersection(const Point& a1, const Point& a2, const Point& b1, const Point& b2) {
    double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - a1),
           c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}

bool OnSegment(const Point& p, const Point& a1, const Point& a2) {
```

```
    return !dcmp(Cross(a1 - p, a2 - p)) && dcmp(Dot(a1 - p, a2 - p)) < 0;
}

double PolygonArea(Point* p, int n) {
    double area = 0;
    for(int i = 1; i < n - 1; i++)
        area += Cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}

void Convex(Point *a, Point *b, int n){
    sort(a+1, a+1+n);
    int top = 0;
    for(int i = 1; i <= n; i++) {
        while(top > 1 && dcmp((a[i] - b[top-1]) ^ (b[top] - b[top-1])) >= 0)
            top--;
        b[++top] = a[i];
    }
    int k = top;
    for(int i = n-1; i > 0; i--) {
        while(top > k && dcmp((a[i] - b[top-1]) ^ (b[top] - b[top-1])) >= 0)
            top--;
        b[++top] = a[i];
    }
}

Point ReadPoint(){
    Point res;
    scanf("%lf%lf", &res.x, &res.y);
    return res;
}

struct Line
{
    Point fr, to;
    double Ang;
    Line(){}
    Line(Point a, Point b, double ang):fr(a), to(b), Ang(ang){}
    bool operator < (const Line&rhs) const {
        return dcmp(Ang - rhs.Ang) < 0;
    }
} l[maxn];

bool cmp(const int &a, const int &b) {
    if (dcmp(l[a].Ang - l[b].Ang) == 0) return dcmp((l[b].fr-l[a].fr)^(l[b].to-l[a].fr)) > 0;
    return dcmp(l[a].Ang - l[b].Ang) < 0;
}

int deq[maxn];
int index[maxn];

Point getIntersection(Line &a, Line &b) {
    double dot1 = Cross(a.to-b.fr, a.fr-b.fr);
    double dot2 = Cross(b.to-a.to, a.fr-a.to);
    double x = (b.fr.x*dot2+b.to.x*dot1)/(dot2+dot1);
    double y = (b.fr.y*dot2+b.to.y*dot1)/(dot2+dot1);
    return Point(x, y);
}

bool check(Line &a, Line &b, Line &c) {
    Point p = getIntersection(b, c);
    return dcmp((a.fr-p)^(a.to-p)) < 0;
}

int HalfPlaneIntersection(int n, Point *b) {
    for (int i = 1; i <= n; i++) index[i] = i;
    sort(index + 1, index + 1 + n, cmp);
    for (int i = 1, j = 0; i <= n; i++) {
        if(i == n) n = j;
        if (j && dcmp(l[index[i]].Ang - l[index[j]].Ang) == 0) continue;
        index[++j] = index[i];
        if(j > n) n = j;
    }
    int head = 1, tail = 0;
    deq[0] = index[1], deq[1] = index[2];
    for (int i = 3; i <= n; i++) {
        while (tail < head && check(l[index[i]], l[deq[head-1]], l[deq[head]])) head--;
        while (tail < head && check(l[index[i]], l[deq[tail+1]], l[deq[tail]])) tail++;
        deq[++head] = index[i];
```

```
    }
    while (tail < head && check(l[deq[tail]], l[deq[head-1]], l[deq[head]])) head--;
    while (tail < head && check(l[deq[head]], l[deq[tail+1]], l[deq[tail]])) tail++;
    deq[++head] = deq[tail];
    for (int i = 0; i + tail < head; i++)
        b[i] = getIntersection(l[deq[i+tail+1]], l[deq[i+tail]]);
    return head-tail;
}
```

半平面交

```
//最后点放在p[m]
// 半平面交
p[maxn];
int m;
inline bool zero(double a) {return abs(a) < eps;}
struct Segment {
    Point s, e;
    double angle;
    void getAngle(){angle = atan2(e.y - s.y, e.x - s.x);}
    //偏移dis距离
    void change(double dis) {
        double len = (e - s).getDis();
        double dx = (s.y - e.y) / len * dis;
        double dy = (e.x - s.x) / len * dis;
        s = s + Point(dx, dy);
        e = e + Point(dx, dy);
    }
}seg[maxn];
Point getIntersect(Segment s1, Segment s2){
    double u = (s1.e - s1.s) ^ (s2.s - s1.s);
    double v = (s1.s - s1.e) ^ (s2.e - s1.e);
    Point t;
    t.x = (s2.s.x * v + s2.e.x * u) / (u + v);
    t.y = (s2.s.y * v + s2.e.y * u) / (u + v);
    return t;
}
bool cmp(Segment s1, Segment s2){
    //先按极角排序
    if(s1.angle < s2.angle - eps) return true;
    //极角相等，内侧的在前
    else if(zero(s1.angle - s2.angle) && ((s2.e - s2.s) ^ (s1.e - s2.s)) > -eps) return true;
    return false;
}
Segment deq[maxn];
// 逆时针把线段加入到seg里面
void HalfPlaneIntersect(Segment seg[], int n){
    sort(seg, seg + n, cmp);
    int tmp = 1;
    for(int i = 1; i < n; ++i)
        if(!zero(seg[i].angle - seg[tmp-1].angle))
            seg[tmp++] = seg[i];
    n = tmp;
    deq[0] = seg[0]; deq[1] = seg[1];
    int head = 0, tail = 1;
    for(int i = 2; i < n; ++i){
        while(head < tail && ((seg[i].e - seg[i].s) ^ (getIntersect(deq[tail],deq[tail-1]) - seg[i].s)) < -eps) tail--;
        while(head < tail && ((seg[i].e - seg[i].s) ^ (getIntersect(deq[head],deq[head+1]) - seg[i].s)) < -eps) head++;
        deq[++tail] = seg[i];
    }
    while(head < tail && ((deq[head].e - deq[head].s) ^ (getIntersect(deq[tail],deq[tail-1]) - deq[head].s)) < -eps) tail--;
    while(head < tail && ((deq[tail].e - deq[tail].s) ^ (getIntersect(deq[head],deq[head+1]) - deq[tail].s)) < -eps) head++;
    if(head == tail) return;
    m = 0;
    for(int i = head; i < tail; ++i)
        p[m++] = getIntersect(deq[i], deq[i+1]);
    if(tail > head + 1)
        p[m++] = getIntersect(deq[head], deq[tail]);
}
double getArea(Point p[], int &n){
    double area = 0;
    for(int i = 1; i < n - 1; ++i)
        area += (p[i] - p[0]) ^ (p[i+1] - p[0]);
    return fabs(area) / 2.0;
}
```

凸包，点在平面内，点集直线左边

```
int Gh(Point *p, int tot) {
    sort(p, p + tot);
    int n = 0;
    for(int i = 0; i < tot; i++) {
        while(n > 1 && ((ploy[n-1] - ploy[n-2]) ^ (p[i] - ploy[n-1])) <= 0) n--;
        ploy[n++] = p[i];
    }
    int m = n;
    for(int i = tot - 2; ~i; i--) {
        while(m > n && ((ploy[m-1] - ploy[m-2]) ^ (p[i] - ploy[m-1])) <= 0) m--;
        ploy[m++] = p[i];
    }
    return m - 1;
}
bool onSegment(Point a, Point b, Point p) {
    return comp((a-p)*(b-p)) <= 0 && comp((a-p)^(b-p)) == 0;
}
int isPointInPolygon(Point p, vector <Point>& poly) {
    int wn = 0;
    int n = poly.size();
    for(int i = 0; i < n; i++) {
        if(onSegment(poly[i], poly[(i+1)%n], p)) return -1; //在边界上
        int k = comp(((poly[(i+1)%n] - poly[i]) ^ (p - poly[i])));
        int d1 = comp(poly[i].y - p.y);
        int d2 = comp(poly[(i+1)%n].y - p.y);
        if(k > 0 && d1 <= 0 && d2 > 0) wn++;
        if(k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    if(wn != 0) return 1; //在内部
    return 0; //在外部
}
bool PolyOnLeft(Point p, Point q, vector <Point>& poly) {
    Vector v = q - p;
    for(int i = 0; i < poly.size(); i++) {
        if(comp(v ^ (poly[i] - p)) < 0) return false;
    }
    return true;
}
```

三角形，旋转卡壳

```
ll getMost(Point* ploy, int n) {
    if(n == 2) return (ploy[1] - ploy[0]).dis2();
    ploy[n] = ploy[0];
    int opa = 1;
    ll ans = 0;
    for(int i = 0; i < n; i++) {
        while(((ploy[opa] - ploy[i]) ^ (ploy[i+1] - ploy[i]))
            < ((ploy[(opa+1)%n] - ploy[i]) ^ (ploy[i+1] - ploy[i])))
            opa = (opa + 1) % n;
        ans = max(ans, (ploy[opa] - ploy[i]).dis2());
        // i与opa是对重点
        // printf("%d  %d\n", i, opa);
    }
    return ans;
}
```