计算几何基础

基本运算

基础类

```
struct Vector {
  double x, y;
  double len() {
      return sqrt(x * x + y * y);
  }
}
Vector operator+(const Vector &r, const Vector &s) {
    return Vector(r.x + s.x, r.y + s.y);
}
Vector operator-(const Vector &r, cosnt Vector &s) {
    return Vector(r.x - s.x, r.y - s.y);
}
Vector opeator*(const Vector &r, double s) {
    return Vector(r.x * s, r.y *s);
}
Vector operator/(const Vector &r, double s) {
    return Vector(r.x / s, r.y / s);
}
```

点积

```
double dot(const Vector &r, const Vector &s) {
    return r.x * s.x + r.y * s.y;
}
```

叉积

```
double cross(const Vector &r, const Vector &s) {
    return r.x * s.y - r.y * s.x;
}
```

线

直线与线段的表示

直线一般通过其上一个点$\boldsymbol{P}$及其方向向量$\vec{\boldsymbol{v}}$来表示。

直线求交

假如有两个直线：$(\boldsymbol{P},\vec{\boldsymbol{u}})$，$(\boldsymbol{Q},\vec{\boldsymbol{v}})$，我们想求其交点，我们设交点$\boldsymbol{S}=\boldsymbol{P}+\boldsymbol{t}\vec{\boldsymbol{u}}$，那么用面积相等列方程解出$\boldsymbol{t}$即可。

```
Point line_intersect(Point P, Vector u, Point Q, Vector v) {
    double t = cross(Q - P, v) / cross(u, v);
    return P + u * t;
}
```

线段判交

跨立实验。

```
bool seg_intersect(Point A, Point B, Point C, Point D) {
    double c1 = cross(Line(A,B),Line(A,C)), c2 = cross(Line(A,B),Line(A,D));
    double c3 = cross(Line(C,D),Line(C,A)), c4 = cross(Line(C,D),Line(C,B));
    return sign(c1) * sign(c2) < 0 && sign(c3) * sign(c4) < 0;
}
```

多边形

求多边形的重心

先三角剖分，然后分别求三角形的重心，按面积加权。

判定点在多边形内

```
bool point_on_line(const Point &p, const Point &A, const Point &B) {
    return sign(cross(B-A, p-A)) == 0 && sign(dot(A-p,B-p)) <= 0;
}
bool in_polygon(const Point &p, const vector<Point> &poly) {
    int n = (int)poly.size();
    int counter = 0;
    for (int i = 0; i < n; ++i) {
        Point a = poly[i], b = poly[(i + 1) % n];
        if (point_on_line(p, a, b)) return true; // bounded included
        int x = sign(cross(p - a, b - a));
        int y = sign(a.y - p.y);
        int z = sign(b.y - p.y);
        if (x > 0 && y <= 0 && z > 0) counter++;
        if (x < 0 && z <= 0 && y > 0) counter--;
    }
    return counter != 0;
}
```

求多边形的面积

```
double area(const vector<Point> &poly) {
    double rt = 0.0;
    int n = (int)poly.size();
    for(int i = 0; i < n; i++) {
        rt += cross(poly[i], poly[(i+1)%n]);
    }
    return fabs(rt / 2.0);
}
```

常用算法

凸包

```
vector<Point> convex(vector<Point> pts) {
    int n = (int)pts.size();
    int m = 0;
    vector<Point> cvx;
    sort(pts.begin(), pts.end());
    for(int i = 0; i < n; i++) {
        while(m > 1 && onleft(cvx[m - 2], cvx[m - 1], pts[i])) {
            cvx.pop_back();
            m--;
        }
        cvx.push_back(pts[i]);
        m++;
    }
    int k = m;
    for(int i = n - 2; i >= 0; i--) {
        while(m > k && onleft(cvx[m - 2], cvx[m - 1], pts[i])) {
            cvx.pop_back();
            m--;
        }
        cvx.push_back(pts[i]);
        m++;
    }
    if(n > 1) {
        m--;
        cvx.pop_back();
    }
    return cvx;
}
```

半平面交

```
struct Line {
    Point p;
    Vector u;
    double ang;
    Line(){}
    Line(Point p, Vector u):p(p),u(u),ang(u.ang()){}
};
bool operator<(const Line &r, const Line &s) {
    return r.ang < s.ang;
}
bool operator ==(const Line &r, const Line &s) {
    return sign(r.ang - s.ang) == 0;
}

bool onleft(const Point &a, const Point &b, const Point &p) {
    return cross(b - a,p - a) > 0;
}
bool onleft(const Line &l, const Point &p) {
    return cross(l.u,p-l.p) > 0;
}
Point line_intersect(Point P, Vector u, Point Q, Vector v) {
    Vector w = P - Q;
    double t = cross(v,w) / cross(u,v);
    return P + u * t;
}
Point line_intersect(const Line &a, const Line &b) {
    return line_intersect(a.p,a.u,b.p,b.u);
}
int half_plane_intersect(vector<Line> &h) {
    sort(h.begin(),h.end());

    int n = (int)h.size();
    int first, last;
    Point *p = new Point[n];
    Line *q = new Line[n];
    q[first = last = 0] = h[0];
    for(int i = 1; i < n; i++) {
        while(first < last && !onleft(h[i],p[last-1])) last--;
        while(first < last && !onleft(h[i],p[first])) first++;
        q[++last] = h[i];
        if(fabs(cross(q[last].u,q[last-1].u)) < eps) {
            last--;
            if(onleft(q[last],h[i].p)) q[last] = h[i];
        }
        if(first < last) p[last-1] = line_intersect(q[last-1],q[last]);
    }
    while(first < last && !onleft(q[first],p[last-1])) last--;
    if(last - first <= 1) return 0;
    p[last] = line_intersect(q[last], q[first]);
    return last - first + 1;
}
```