点和线程序表示

```
// 二维坐标
struct point{
  double x, y;
};

// 两点确定一条直线
struct line{
  point a,b;
};
```

点积与叉积

点积:

$$\vec{a} = (x_1, y_1), \vec{b} = (x_2, y_2)$$
$$\vec{a} * \vec{b} = |\vec{a}||\vec{b}|cos\alpha = x_1 * x_2 + y_1 * y_2$$

点积的应用:

1. 计算夹角
2. 检测正交性，两个向量正交，点积为0
3. 判断向量方向:
   - 点积值 > 0 , 方向基本相同，夹角在 `0~90`
   - 点积值 == 0， 正交相互垂直
   - 点积值 < 0, 方向相反，夹角在 `90~180`

```
/**
 * @brief 点积
 * @param  p1
 * @param  p2
 * @param  p0
 * @return double
 * */
double dmult(point p1, point p2, point p0) {
  return (p1.x - p0.x) * (p2.x - p0.x) + (p1.y - p0.y) * (p2.y - p0.y);
}
```

叉积:

$$\vec{a} = (x_1, y_1), \vec{b} = (x_2, y_2)$$
$$\vec{a} \times \vec{b} = (x_1 * y_2) - (x_2 * y_1)$$

叉积的应用:

1. 多用于计算两个向量构成的平行四边形面积
2. 计算两个向量的关系
   - $\vec{a} \times \vec{b} > 0$ 表示$\vec{a}$在$\vec{b}$之上，即 $\vec{a}$在$\vec{b}$的上半部分的逆时针的 `0~180` ；
   - $\vec{a} \times \vec{b} == 0$ 表示$\vec{a}$与$\vec{b}$同向或逆向;
   - $\vec{a} \times \vec{b} < 0$ 表示$\vec{a}$在$\vec{b}$之下, 即 $\vec{a}$在$\vec{b}$的下半部分的顺时针的 `0~180` 。

```
/**
 * @brief 叉积
 * @param  p1
 * @param  p2
 * @param  p0
 * @return double
 * */
double xmult(point p1, point p2, point p0) {
  return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}
```

多边形判断

凸多边形

凸多边形（ `Convex Polygon` ）指如果把一个多边形的所有边中，任意一条边向两方无限延长成为一直线时，其他各边都在此直线的同旁.

1. 允许顶点共线
   利用一个 `bool` 数组暂存临边的叉积计算结果, 以为叉积的值只有 `>0,=0,<0` 三种情况，所以该数组的大小为 `3` 即可。
   如果为凸多边形,叉积的计算结果全部大于0,小于0。
   若出现大于0,小于0同时存在则必不为凸多边形。

```
/**
 * @brief 判断凸多边形，顶点顺时针或逆时针给出
 *        根据定义，选中一条边其余边均在，这条边的同侧；
 *        即对每条边与临边的叉积的结果均大于0(顺时针给出)，或小于0(逆时针给出)，或==0(允许顶点共线)
 * @param  points
 * @param  n
 * @return int
 * */
int ConvexPolygon(int n, point *points) {
  int i, s[3] = {1, 1, 1};
  for (i = 0; i < n && s[1] | s[2]; i++) {
    s[_sign(xmult(points[(i + 1) % n], points[(i + 2) % n], points[i]))] = 0;
  }

  return s[1] | s[2];
}
```

2. 不允许相邻边共线

   与上述算法类似，但如果两个向量共线，其叉积为0，此时 `s[0] = 0` ,存在共线顶点，直接返回即可。

```
/**
 * @brief 判断凸多边形，顶点顺时针或逆时针给出，不允许共线
 *        根据定义，选中一条边其余边均在，这条边的同侧；
 *        即对每条边与临边的叉积的结果均大于0，或小于0，或==0(允许顶点共线)
 * @param  points
 * @param  n
 * @return int
 * */
int ConvexPolygon_2(int n, point *points) {
  int i, s[3] = {1, 1, 1};
  for (i = 0; i < n && s[0] && s[1] | s[2]; i++) {
    s[_sign(xmult(points[(i + 1) % n], points[(i + 2) % n], points[i]))] = 0;
  }

  return s[0] && s[1] | s[2];
}
```

点与多边形关系

1. 点在凸多边形内，允许点在多边形上

```
/**
 * @brief 判断点在多边形内，允许点在多边形上。
 *        处理逻辑与凸多边形判断类似。判断点q与多边形的顶点构成的向量，
 *        与多边形边的向量的叉积，如果在多边形内或在多边形上，其叉积一定全部>0,或全部<0
 * @param  q
 * @param  n
 * @param  points
 * @return int
 * */
int insideConvex(point q, int n, point *points) {
  int i, s[3] = {1, 1, 1};
  for (i = 0; i < n && s[1] | s[2]; i++) {
    s[_sign(xmult(points[(i + 1) % n], q, points[i]))] = 0;
  }

  return s[1] | s[2];
}
```

2. 点在凸多边形内，不允许点在多边形上

```
/**
 * @brief 判断点在多边形内，不允许点在多边形上。
 *        处理逻辑与凸多边形判断类似。判断点q与多边形的顶点构成的向量，
 *        与多边形边的向量的叉积，如果在多边形内或在多边形上，其叉积一定全部>0,或全部<0
 * @param  q
 * @param  n
 * @param  points
 * @return int
 * */
int insideConvex_2(point q, int n, point *points) {
  int i, s[3] = {1, 1, 1};
  for (i = 0; i < n && s[0] && s[1] | s[2]; i++) {
    s[_sign(xmult(points[(i + 1) % n], q, points[i]))] = 0;
  }

  return s[0] && s[1] | s[2];
}
```

3. 点在多边形内或在多边形上

```c
/**
 * @brief 判断点在任意多边形内，允许在多边形上.顶点顺时针或逆时针给出
 *        从目标点p出发引出一条射线，计算这条射线与多边形的交点个数
 *        如果在多边形内，其交点个数必为奇数，否在为偶数
 * @param  q
 * @param  n
 * @param  points
 * @param  on_edge 点是否在多边形上
 * @return int
 * */
int insidePlogon(point q, int n, point *points, int on_edge = 1) {
  point q2;  // 代表了无穷远处的一个顶点，与目标点p构成了一条射线。
  q2.x  = rand() + offset;
  q2.y  = rand() + offset;
  int i = 0, count = 0;

  while (i < n) {
    for (i = 0; i < n; i++) {
      if (zero(xmult(q, points[i], points[(i + 1) % n])) &&
          (points[i].x - q.x) * (points[(i + 1) % n].x - q.x) < eps &&
          (points[i].y - q.y) * (points[(i + 1) % n].y - q.y) < eps) {
        // 如果点在多边形上，其与顶点构成的向量中必定存在叉积为为0的情况，zero函数在为0是返回1
        // 在叉积为0的情况下需要判断，p是否在两个顶点构成的线段的延长线上，在延长线上点不会出现在
        // 多边形上，而是多边形外。
        // 判断p是否在线段内，只需判断q的坐标大于期中一个顶点的坐标，小于另一个顶点的坐标
        return on_edge;
      } else if (zero(xmult(q, q2, points[i]))) {
        // 当存在一个顶点在目标点为起始的线段上，说明p要么在多边形内，要么在多边形外
        // 此时可以终止循环判断
        break;
      } else if (xmult(q, points[i], q2) * xmult(q, points[(i + 1) % n], q2) <
                     -eps &&
                 xmult(points[i], q, points[(i + 1) % n]) *
                         xmult(points[i], q2, points[(i + 1) % n]) <
                     -eps) {
        // 判断射线与顶点构成的线段是否相交。
        count++;
      }
    }
  }

  return count & 1;
}

/**
 * @brief
 * 判断两条线段是否相交，如果两条直线相交，对应的端点与另一条线段的叉积的乘积一定小于0
 *        如果不想交，两者的叉积乘积一定大于0. 当两个线段共线叉积的乘积等于0
 * @param  l1
 * @param  l2
 * @param  p1
 * @param  p2
 * @return int
 * */
```

```
int oppositeSide(point l1, point l2, point p1, point p2) {
  return xmult(l1, p1, l2) * xmult(l1, p2, l2) < -eps;
}

/**
 * @brief 判断点是否在线段内，如果在线段内，点与线段端点的构成的向量，叉积==0
 *        同时该点的坐标在线段的端点坐标之间。
 * @param  l1
 * @param  l2
 * @param  p
 * @return int
 * */
int dotOnLineIn(point l1, point l2, point p) {
  return zero(xmult(l1, p, l2)) && (l1.x - p.x) * (l2.x - p.x) < eps &&
         (l1.y - p.y) * (l2.y - p.y) < eps;
}
```

线段与多边形的关系

```c
/**
 * @brief 判断线段是否在多边形内
 * @param  l1
 * @param  l2
 * @param  n
 * @param  points
 * @return int
 * */
int insidePolygon(point l1, point l2, int n, point *points) {
  point t[MAXN], tt;
  int   i, j, k = 0;

  // 如果端点不在多边形内，那么线段一定不在多边形内
  if (!insidePlogon(l1, n, points) || !insidePlogon(l2, n, points)) {
    return 0;
  }

  for (int i = 0; i < n; i++) {
    // 如果线段和一条边相交返回0,内交
    if (oppositeSide(l1, l2, points[i], points[(i + 1) % n]) &&
        oppositeSide(points[i], points[(i + 1) % n], l1, l2)) {
      return 0;
    } else if (dotOnLineIn(l1, points[i], points[(i + 1) % n])) {
      t[k++] = l1;  // l1 在多边形边上
    } else if (dotOnLineIn(l2, points[i], points[(i + 1) % n])) {
      t[k++] = l2;  // l2 在多边形边上
    } else if (dotOnLineIn(points[i], l1, l2)) {
      t[k++] = points[i];  // 顶点在线段上
    }
  }

  for (i = 0; i < k; i++) {
    for (j = i + 1; j < k; j++) {
      tt.x = (t[i].x + t[j].x) / 2;  // 求两个交点的中点
      tt.y = (t[i].y + t[j].y) / 2;
      if (!insidePlogon(
              tt,
              n,
              points)) {  // 如果交点的中点不在多边形内，线段一定不在多边形内
        return 0;
      }
    }
  }

  return 1;
}
```

求两个线段的交点

设二维向量$\overrightarrow{p_1} = (x_1, y_2), \overrightarrow{p_2} = (x_2, y_2), \overrightarrow{q_1} = (a_1, b_1), \overrightarrow{q_2} = (a_2, b_2)$，求两直线的交点：
利用变量 t 将直线$\overrightarrow{p_1} - \overrightarrow{p_2}$上的点表示为$\overrightarrow{p_1} + t(\overrightarrow{p_2} - \overrightarrow{p_1})$，又因为交点落在直线$\overrightarrow{q_2} - \overrightarrow{q_1}$上。
所以有：

$$(\overrightarrow{q_2}-\overrightarrow{q_1})\times(\overrightarrow{p_1}+t(\overrightarrow{p_2}-\overrightarrow{p_1})-\overrightarrow{q_1})=0$$

转换为行列式计算为：

$$(\begin{bmatrix}x_1\\y_1\end{bmatrix}+t(\begin{bmatrix}x_2-x_1\\y_2-y_1\end{bmatrix})-\begin{bmatrix}a_1\\b_1\end{bmatrix})\times\begin{bmatrix}a_2-a_1\\b_2-b_1\end{bmatrix}=0$$

可以得出：

$$t*((a_2-a_1)(y_1-y_2)-(b_2-b_1)(x_1-x_2))=(b_2-b_1)(x_1-a_1)-(a_2-a_1)(y_1-y2)$$

最后推到出：

$$t=\frac{(b_2-b_1)(x_1-a_1)-(a_2-a_1)(y_1-y_2)}{(a_2-a_1)(y_1-y_2)-(b_2-b_1)(x_1-x_2)}$$

最后得出交点为：

$$\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}x_1\\y_1\end{bmatrix}+\frac{(b_2-b_1)(x_1-a_1)-(a_2-a_1)(y_1-y_2)}{(a_2-a_1)(y_1-y_2)-(b_2-b_1)(x_1-x_2)}*\begin{bmatrix}x_2-x_1\\y_2-y_1\end{bmatrix}$$

```
/**
 * @brief 求两条直线的交点
 * @param  u
 * @param  v
 * @return point
 * */
point intersection(line u, line v) {
  point  ret = u.a;
  double t =
      ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) * (v.a.x - v.b.x)) /
      ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));

  ret.x += (u.b.x - u.a.x) * t;
  ret.y += (u.b.y - u.b.y) * t;

  return ret;
}
```

重心

1. 三角形重心

```
/**
 * @brief 求三角形的重心
 *        三角形两条中线的交点，即为重心
 * @param  p1
 * @param  p2
 * @param  p3
 * @return point
 * */
point barycenter(point p1, point p2, point p3) {
  line u, v;
  u.a.x = (p1.x + p2.x) / 2;
  u.a.y = (p1.y + p2.y) / 2;
  u.b   = p3;

  v.a.x = (p1.x + p3.x) / 2;
  v.a.y = (p1.y + p3.y) / 2;
  v.b   = p2;

  return intersection(u, v);
}
```

2. 多边形重心

平面多边形X可以被分解为n个简单图形$x_1, x_2, x_3, \ldots, x_n$,这些简单图形的重心为$C_i$, 简单图形面积为$X_i$, 则这个平面多边形的重心坐标为$(C_x, C_y)$,有：

$$C_x = \frac{\sum C_{ix} X_i}{\sum A_i}$$
$$C_y = \frac{C_{iy} X_i}{\sum X_i}$$

多边形重心横坐标 = 多边形剖分的每一个三角形重心的横坐标 * 该三角形的面积之和 / 多边形总面积

多边形重心纵坐标 = 多边形剖分的每一个三角形重心的纵坐标 * 该三角形的面积之和 / 多边形总面积

多边形面积计算可以简化为：

$$S = \frac{\overrightarrow{PA} \times \overrightarrow{PB}}{2}$$

```
/**
 * @brief 多边形重心
 * @param  n
 * @param  points
 * @return point
 * */
point barycenter(int n, point *points) {
  point  ret, t;
  double t1 = 0, t2;
  int    i;
  ret.x = ret.y = 0;

  for (int i = 1; i < n - 1; i++) {
    if (fabs(t2 = xmult(points[0], points[i], points[i + 1])) > eps) {
      t = barycenter(points[0], points[i], points[i + 1]);
      ret.x += t.x * t2;
      ret.y += t.y * t2;
      t1 += t2;
    }
  }

  if (fabs(t1) > eps) {
    ret.x /= t1;
    ret.y /= t1;
  }

  return ret;
}
```

线段不相交

```
// 判断线段是否不相交
int sameSide(point p1, point p2, point l1, point l2) {
  return xmult(l1, p1, l2) * xmult(l1, p2, l2) > eps;
}
```

多边形切割：

```
/**
 * @brief  多边形沿l1l2,在silde侧切割
 * @param  n
 * @param  p
 * @param  l1
 * @param  l2
 * @param  silde
 * */
void ploygonCut(int &n, point *p, point l1, point l2, point silde) {
  point pp[100];
  int   m = 0, i;

  for (i = 0; i < n; i++) {
    // 如果p[i]silde构成的直线与l1l2不相交，直接加入到pp中
    if (sameSide(p[i], silde, l1, l2)) {
      pp[m++] = p[i];
    }

    // 临边与l1l2相交的情况，加入其交点
    if (!sameSide(p[i], p[(i + 1) % n], l1, l2) &&
        !(zero(xmult(p[i], l1, l2))) && (zero(xmult(p[(i + 1) % n], l1, l2)))) {
      pp[m++] = intersection(p[i], p[(i + 1) % n], l1, l2);
    }
  }

  // 去除坐标重叠的部分，
  for (n = i = 0; i < m; i++) {
    if (!i || !zero(pp[i].x - pp[i - 1].x) || !zero(pp[i].y - pp[i - 1].y)) {
      p[n++] = pp[i];
    }
  }

  // 判断首位端点是否相同，如果相同则丢弃
  if (zero(p[n - 1].x - p[0].x) && zero(p[n - 1].y - p[0].y)) {
    n--;
  }

  // 不构成多边形
  if (n < 3) {
    n = 0;
  }
}
```