前置知识点

1. 如何求 π：`pi = acos(-1)`
2. 余弦定理：$c^2 = a^2 + b^2 - 2ab\cos t$

浮点数的比较

由于计算机里存储的小数，都是近似值，因此比较的时候，需要避免该误差引发的错误

```
const double eps = 1e-8;
int sign(double x)  //符号函数
{
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
int cmp(double x, double y) //比较函数
{
    if (fabs(x - y) < eps) reurn 0;
    if (x < y) return -1;
    return 1;
}
```

向量

向量的基本运算

向量的 **加、减、数乘** 运算

向量的点乘（内积）

$$ \bold{A} \cdot \bold{B} = |\bold{A}||\bold{B}|\cos \theta $$

几何意义：向量 $\bold{A}$ 在向量 $\bold{B}$ 上的投影与向量 $\bold{B}$ 的长度的乘积

代码实现：

```
double dot(Point a, Point b)
{
    return a.x * b.x + a.y * b.y;
}
```

向量的叉乘（外积）

$$ \bold{A} \times \bold{B} = |\bold{A}| |\bold{B}| \sin \theta $$

几何意义：向量 $\bold{A}$ 与向量 $\bold{B}$ 张成的平行四边形的有向面积 ($\bold{B}$ 在 $\bold{A}$ 的逆时针方向为正)

代码实现：

```
double cross(Point a, Point b)
{
    return a.x * b.y - b.x * a.y;
}
```

常用函数

求向量模长

$$ |\bold{A}| = \sqrt{\bold{A} \cdot \bold{A}} $$

```
double get_length(Point a)
{
    return sqrt(dot(a, a));
}
```

计算向量夹角

$$ \cos \theta = \dfrac{\bold{A} \cdot \bold{B}}{|\bold{A}||\bold{B}|} $$

```
double get_angle(Point a, Point b)
{
    return acos(dot(a, b) / get_length(a) / get_length(b));
}
```

计算两个向量构成的平行四边形有向面积

$$ S_{ABCD} = \vec{A} \times \vec{B} $$

```
double area(Point a, Point b, Point c)
{
    return cross(b - a, c - a);
}
```

向量A顺时针旋转C的角度：

$$ \begin{pmatrix} x' & y' \end{pmatrix} = \begin{pmatrix} x & y \end{pmatrix} \cdot \begin{pmatrix} \cos \theta & -\sin \theta \ \sin \theta & \cos \theta \end{pmatrix} $$

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

点与线

常见直线表示形式

一般式 $ax + by + c = 0$

点向式 $p_0 + \bold{v} t$

斜截式 $y = kx + b$

常用操作

判断点在直线上

$$ \bold{A \times B = 0} $$

两直线相交的交点

```
// cross(v, w) == 0则两直线平行或者重合
Point get_line_intersection(Point p, Vector v, Point q, vector w)
{
    vector u = p - q;
    double t = cross(w, u) / cross(v, w);
    return p + v * t;
}
```

点到直线的距离

$$ S_{ABCD} = dh = |b - a| \cdot h $$

```
double distance_to_line(Point p, Point a, Point b)
{
    vector v1 = b - a, v2 = p - a;
    return fabs(cross(v1, v2) / get_length(v1));
}
```

点到线段的距离

分类讨论边界情况

```
double distance_to_segment(Point p, Point a, Point b)
{
    if (a == b) return get_length(p - a);
    Vector v1 = b - a, v2 = p - a, v3 = p - b;
    if (sign(dot(v1, v2)) < 0) return get_length(v2);
    if (sign(dot(v1, v3)) > 0) return get_length(v3);
    return distance_to_line(p, a, b);
}
```

点在直线上的投影

```
double get_line_projection(Point p, Point a, Point b)
{
    Vector v = b - a;
    return a + v * (dot(v, p - a) / dot(v, v));
}
```

点是否在线段上

共线且不在两侧

```
bool on_segment(Point p, Point a, Point b)
{
    return sign(cross(p - a, p - b)) == 0 && sign(dot(p - a, p - b)) <= 0;
}
```

判断两线段是否相交

基于跨立实验

```
bool segment_intersection(Point a1, Point a2, Point b1, Point b2)
{
    double c1 = cross(a2 - a1, b1 - a1), c2 = cross(a2 - a1, b2 - a1);
    double c3 = cross(b2 - b1, a2 - b1), c4 = cross(b2 - b1, a1 - b1);
    return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0;
}
```

多边形

三角形

面积

1. 叉乘（坐标）

$$ S = \dfrac{1}{2} \bold{A} \bold{B} $$

2. 海伦公式（边长）

$$ \text{令 } p = \dfrac{1}{2}(a+b+c)\text{，则 } S = \sqrt{p(p-a)(p-b)(p-c)} $$

三角形四心

1. 外心，外接圆圆心

三边中垂线交点

到三角形三个顶点的距离相等

2. 内心，内切圆圆心

角平分线交点

到三边距离相等

3. 垂心

三条垂线交点

4. 重心

三条中线交点

到三角形三顶点距离的平方和最小的点，三角形内到三边距离之积最大的点

普通多边形

> 通常按逆时针存储所有点

定义

多边形：由在同一平面且不再同一直线上的多条线段首尾顺次连接且不相交所组成的图形叫多边形

简单多边形：简单多边形是除相邻边外其它边不相交的多边形

凸多边形：过多边形的任意一边做一条直线，如果其他各个顶点都在这条直线的同侧，则把这个多边形叫做凸多边形

任意凸多边形外角和均为 $360°$

任意凸多边形内角和为 $(n−2) \times 180°$

常用函数

求多边形面积（不一定是凸多边形）

我们可以从第一个顶点处把凸多边形分成 $n - 2$ 个三角形，然后把 **有向面积** 加起来（三角剖分）

```
double polygon_area(Point p[], int n)
{
    double s = 0;
    for (int i = 1; i + 1 < n; i ++ )
        s += cross(p[i] - p[0], p[i + 1] - p[i]);
    return s / 2;
}
```

判断点是否在多边形内（不一定是凸多边形）

- 射线法：从该点任意做一条和所有边都不平行的射线。交点个数为偶数，则在多边形外，为奇数，则在多边形内。
- 转角法

判断点是否在凸多边形内

只需判断点是否在所有 **有向边的左边**（逆时针存储多边形）

皮克定理

皮克定理是指一个计算点阵中顶点在格点上的多边形面积公式该公式可以表示为:

$$ S = a + \dfrac{b}{2} - 1 $$

其中 a 表示多边形内部的点数，b 表示多边形边界上的点数，S 表示多边形的面积。

圆

- 圆与直线交点
- 两圆交点
- 点到圆的切线
- 两圆公切线
- 两圆相交面积

凸包

二维凸包

在平面上能包含所有给定点的最小凸多边形叫做凸包。

其定义为：对于给定集合 X ，所有包含 X 的凸集的交集 S 被称为 X 的 **凸包** 。

实际上可以理解为用一个橡皮筋包含住所有给定点的形态。

凸包用最小的周长围住了给定的所有点。如果一个凹多边形围住了所有的点，它的周长一定不是最小，如下图。根据三角不等式，凸多边形在周长上一定是最优的。

凸包的求法

常用的求法有 Graham 扫描法和 Andrew 算法，这里主要介绍 Andrew 算法。

Andrew 算法求凸包

该算法的时间复杂度为 $O(n\log n)$ ，其中 n 为待求凸包点集的大小，同时复杂度的瓶颈也在于对所有点坐标的双关键字排序。

首先把所有点以横坐标为第一关键字，纵坐标为第二关键字排序。

显然排序后最小的元素和最大的元素一定在凸包上。而且因为是凸多边形，我们如果从一个点出发逆时针走，轨迹总是"左拐"的，一旦出现右拐，就说明这一段不在凸包上。因此我们可以用一个 **单调栈** 来维护上下凸壳。

因为从左向右看，上下凸壳所旋转的方向不同，为了让单调栈起作用，我们首先 **升序** 枚举求出 **下凸壳**，然后 **降序** 枚举求出 **上凸壳**。

求凸壳时，一旦发现即将进栈的点（P）和栈顶的两个点（S_1，S_2 其中 S_1 为栈顶）行进的方向向右旋转，即叉积小于 0：$\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} < 0$，则弹出栈顶，回到上一步，继续检测，直到 $\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} \ge 0$ 或者栈内仅剩一个元素为止。

通常情况下不需要保留位于凸包边上的点，因此上面一段中 $\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} < 0$ 这个条件中的"$\lt$"可以视情况改为 $\le$，同时后面一个条件应改为 $\gt$。

左手进栈 右手出栈

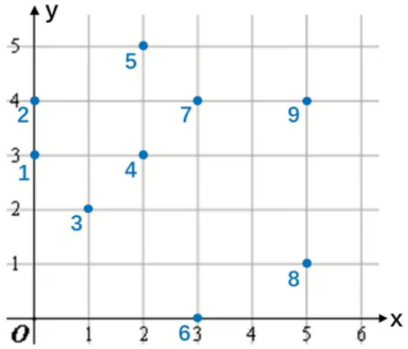
凸包 Andrew算法

Andrew 算法

1. 对所有点按坐标 x 为第一关键字、 y 为第二关键字排序。第1、第n两个点一定在凸包上。
2. 先**顺序枚举**所有点，求**下凸包**。用**栈**维护当前在凸包上的点：新点入栈前，总要判断该弹出哪些旧点。只要新点处在由栈顶两点构成的有向直线的右侧或共线，就弹出旧点。不能弹出时，新点入栈。
3. 再**逆序枚举**所有点，求**上凸包**。用**栈**维护同上。

注意：每个点入栈两次，出栈不超过两次，所以总次数不超过 $4n$。

```
9
0 3
0 4
1 2
2 3
2 5
3 0
3 4
5 1
5 4
```




```
struct Point{double x,y;} p[N],s[N];
int n,top;

double cross(Point a,Point b,Point c){ //叉积
  return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
}
double dis(Point a,Point b){ //距离
  return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
bool cmp(Point a, Point b){ //比较
  return a.x!=b.x ? a.x<b.x : a.y<b.y;
}
double Andrew(){
  sort(p+1,p+n+1,cmp); //排序
  for(int i=1; i<=n; i++){ //下凸包
    while(top>1&&cross(s[top-1],s[top],p[i])<=0)top--;
    s[++top]=p[i];
  }
  int t=top;
  for(int i=n-1; i>=1; i--){ //上凸包
    while(top>t&&cross(s[top-1],s[top],p[i])<=0)top--;
    s[++top]=p[i];
  }
  double res=0; //周长
  for(int i=1; i<top; i++) res+=dis(s[i],s[i+1]);
  return res;
}
```