计算几何

mcfx

2019 年 7 月 5 日

计算几何和解析几何的区别

解析几何中求两条直线交点怎么做?

计算几何和解析几何的区别

解析几何中求两条直线交点怎么做？
——列式子，解方程

计算几何和解析几何的区别

解析几何中求两条直线交点怎么做?
——列式子，解方程
那求两个圆的交点呢?

计算几何和解析几何的区别

解析几何中求两条直线交点怎么做?
——列式子，解方程
那求两个圆的交点呢?
——列式子，解方程

计算几何和解析几何的区别

解析几何中求两条直线交点怎么做?
——列式子，解方程
那求两个圆的交点呢?
——列式子，解方程
。。。那求直线与圆的交点呢?

计算几何和解析几何的区别

解析几何中求两条直线交点怎么做?
——列式子，解方程
那求两个圆的交点呢?
——列式子，解方程
。。。那求直线与圆的交点呢?
——列式子，解方程

计算几何和解析几何的区别

没错，OI 不是数学，采用“解方程”这种需要大量人力的工具既没有办法充分发挥计算机的优势，也会使代码变得十分难看

计算几何和解析几何的区别

没错，OI 不是数学，采用“解方程”这种需要大量人力的工具既没有办法充分发挥计算机的优势，也会使代码变得十分难看
而且要是代码里写了一大坨式子万一 WA 了怎么调……

计算几何和解析几何的区别

没错，OI 不是数学，采用“解方程”这种需要大量人力的工具既没有办法充分发挥计算机的优势，也会使代码变得十分难看
而且要是代码里写了一大坨式子万一 WA 了怎么调……
计算几何在很大程度上避免了繁琐的解方程过程，代码简单便于调试，最主要的是我们可以扔掉我们的草稿纸了……

点和向量

一个点用 x 和 y 两个坐标来表示。

点和向量

一个点用 x 和 y 两个坐标来表示。
由于一个向量也是两个坐标，因此点类同时也是向量的类。

点和向量

一个点用 x 和 y 两个坐标来表示。
由于一个向量也是两个坐标，因此点类同时也是向量的类。
点 $+$ 向量 $=$ 点，点 $-$ 点 $=$ 向量，向量 $\cdot$ 数值 $=$ 向量。

点的模板

```
struct vec
{
        lf x,y;
        vec(){}
        vec(lf a,lf b){x=a,y=b;}
        inline vec&operator+=(const vec&a)
        {
                x+=a.x,y+=a.y;
                return *this;
        }
        inline vec&operator-=(const vec&a)
        {
                x-=a.x,y-=a.y;
                return *this;
        }
        inline vec&operator/=(const lf&a)
        {
                x/=a,y/=a;
                return *this;
        }
        inline vec&operator*=(const lf&a)
```

点的模板

```
{
        x*=a,y*=a;
        return *this;
}
inline vec operator+(const vec&b)const
{
        vec a=*this;a+=b;return a;
}
inline vec operator-(const vec&b)const
{
        vec a=*this;a-=b;return a;
}
inline vec operator/(const lf&b)const
{
        vec a=*this;a/=b;return a;
}
inline vec operator*(const lf&b)const
{
        vec a=*this;a*=b;return a;
}
```

向量的点积和叉积

点积：$v_1 \cdot v_2 = |v_1| \cdot |v_2| \cdot \cos\theta = x_1 \cdot x2 + y1 \cdot y2$。

向量的点积和叉积

点积：$v_1 \cdot v_2 = |v_1| \cdot |v_2| \cdot \cos\theta = x_1 \cdot x2 + y1 \cdot y2$。

叉积：$v_1 \times v_2 = |v_1| \cdot |v_2| \cdot \sin\theta = x_1 \cdot y2 - y1 \cdot x2$。

向量的点积和叉积

点积：$v_1 \cdot v_2 = |v_1| \cdot |v_2| \cdot \cos\theta = x_1 \cdot x2 + y1 \cdot y2$。
叉积：$v_1 \times v_2 = |v_1| \cdot |v_2| \cdot \sin\theta = x_1 \cdot y2 - y1 \cdot x2$。
θ 表示 v_1 和 v_2 间的夹角。对于叉积，可以认为是 v_1 逆时针转多少度可以得到 v_2。

向量的点积和叉积

点积：$v_1 \cdot v_2 = |v_1| \cdot |v_2| \cdot \cos\theta = x_1 \cdot x2 + y1 \cdot y2$。
叉积：$v_1 \times v_2 = |v_1| \cdot |v_2| \cdot \sin\theta = x_1 \cdot y2 - y1 \cdot x2$。
θ 表示 v_1 和 v_2 间的夹角。对于叉积，可以认为是 v_1 逆时针转多少度可以得到 v_2。
如果 v_2 在 v_1 的顺时针方向那么叉积是负数，否则是正数。方向相同（或相反）时是 0。

向量的点积和叉积

点积：$v_1 \cdot v_2 = |v_1| \cdot |v_2| \cdot \cos\theta = x_1 \cdot x2 + y1 \cdot y2$。
叉积：$v_1 \times v_2 = |v_1| \cdot |v_2| \cdot \sin\theta = x_1 \cdot y2 - y1 \cdot x2$。
θ 表示 v_1 和 v_2 间的夹角。对于叉积，可以认为是 v_1 逆时针转多少度可以得到 v_2。
如果 v_2 在 v_1 的顺时针方向那么叉积是负数，否则是正数。方向相同（或相反）时是 0。
用叉积可以快速求出三角形面积：任意两边对应的向量的叉积的绝对值的一半。

向量的点积和叉积

点积：$v_1 \cdot v_2 = |v_1| \cdot |v_2| \cdot \cos\theta = x_1 \cdot x2 + y1 \cdot y2$。
叉积：$v_1 \times v_2 = |v_1| \cdot |v_2| \cdot \sin\theta = x_1 \cdot y2 - y1 \cdot x2$。
θ 表示 v_1 和 v_2 间的夹角。对于叉积，可以认为是 v_1 逆时针转多少度可以得到 v_2。
如果 v_2 在 v_1 的顺时针方向那么叉积是负数，否则是正数。方向相同（或相反）时是 0。
用叉积可以快速求出三角形面积：任意两边对应的向量的叉积的绝对值的一半。

```
        inline lf operator*(const vec&a)const
        {
                return x*a.x+y*a.y;
        }
        inline lf operator%(const vec&a)const
        {
                return x*a.y-y*a.x;
        }
};
```

向量的极角

可以使用 $atan2(y, x)$ 求出。返回值在 $(-\pi, \pi]$，如果需要 $[0, 2\pi)$ 的角度，需要自行特判。

向量的极角

可以使用 $atan2(y,x)$ 求出。返回值在 $(-\pi,\pi]$，如果需要 $[0,2\pi)$ 的角度，需要自行特判。

```
        inline lf a()
        {
                return atan2(y,x);
        }
};
```

向量的极角

可以使用 $atan2(y,x)$ 求出。返回值在 $(-\pi,\pi]$，如果需要 $[0,2\pi)$ 的角度，需要自行特判。

```
        inline lf a()
        {
                return atan2(y,x);
        }
};
```

极角排序时除了可以直接求出极角，也可以判断象限，然后同一象限内按叉积排序。

向量的旋转

$\vec{v'} = (x\cos\alpha - y\sin\alpha, x\sin\alpha + y\cos\alpha)$，其中 α 表示逆时针旋转的弧度。

向量的旋转

$\vec{v'} = (x\cos\alpha - y\sin\alpha, x\sin\alpha + y\cos\alpha)$，其中 α 表示逆时针旋转的弧度。

```
        inline void rot(lf a)
        {
                lf ca=cos(a),sa=sin(a);
                lf nx=x*ca-y*sa;
                y=x*sa+y*ca,x=nx;
        }
};
```

直线的表示

记录直线上一点与直线的方向向量。

直线的表示

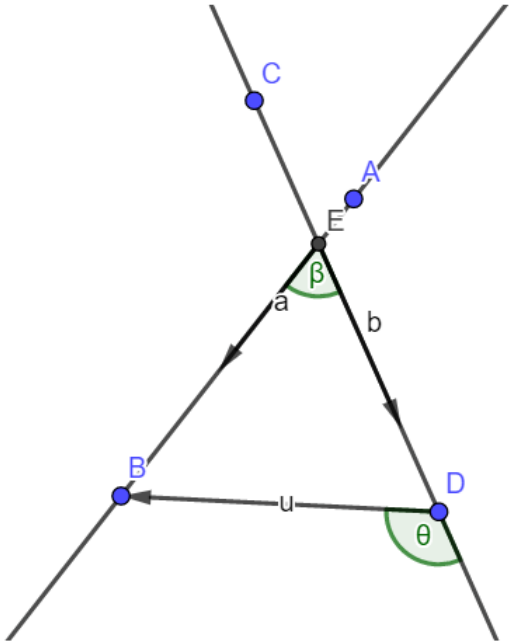
记录直线上一点与直线的方向向量。

```
struct line
{
        vec a,b;
        line(){}
        line(vec x,vec y){a=x,b=y;}
};
```

求两条直线交点

如果两直线相交，则交点只有一个，我们记录了直线上的一个点和直线的方向向量，所以我们只需要知道这个点与交点的距离 l，再将这个点沿方向向量平移 l 个单位长度即可。

考虑构造三角形，利用正弦定理求解 l，可以利用向量积构造出正弦定理。



由上图可知，$|\mathbf{a}\times\mathbf{b}|=|\mathbf{a}||\mathbf{b}|\sin\beta$，$|\mathbf{u}\times\mathbf{b}|=|\mathbf{u}||\mathbf{b}|\sin\theta$。

作商得：

$$T=\frac{|\mathbf{u}\times\mathbf{b}|}{|\mathbf{a}\times\mathbf{b}|}=\frac{|\mathbf{u}|\sin\theta}{|\mathbf{a}|\sin\beta}$$

可以看出，$\left|\frac{|\mathbf{u}|\sin\theta}{\sin\beta}\right|=l$。若绝对值内部式子取值为正，代表沿 $\mathbf{a}$ 方向平移，反之则为反方向。

同时，我们将 T 直接乘上 $\mathbf{a}$，就自动出现了直线的单位向量，不需要进行其他消去操作了。

求两条直线交点

```
inline lf intf(const line&a,const line&b)
{
        return(b.b%(a.a-b.a))/(a.b%b.b);
}

inline vec intsec(const line&a,const line&b)
{
        return a.a+a.b*intf(a,b);
}
```

判断一个点是否在任意多边形内部

从这个点出发引一条射线，如果这条射线与多边形有奇数个交点则在内部，否则在外部。

判断一个点是否在任意多边形内部

从这个点出发引一条射线，如果这条射线与多边形有奇数个交点则在内部，否则在外部。
为了避免射线与多边形上的点重合，可以将射线的斜率设为无理数。

凸包

将凸包分成上凸壳和下凸壳两部分，分别求解。

凸包

将凸包分成上凸壳和下凸壳两部分，分别求解。
每一部分按 x 坐标排序，用单调栈维护，利用叉积的符号判断凹凸性。

凸包

将凸包分成上凸壳和下凸壳两部分，分别求解。
每一部分按 x 坐标排序，用单调栈维护，利用叉积的符号判断凹凸性。
注意排序的时候如果 x 相同，必须以 y 坐标作为第二键值。

Simpson 积分

对于一个二次函数 $f(x)$，有 $\int_l^r f(x)dx = (r-l)\frac{f(l)+4f(\frac{l+r}{2})+f(r)}{6}$。

Simpson 积分

对于一个二次函数 $f(x)$，有 $\int_l^r f(x)dx = (r-l)\frac{f(l)+4f(\frac{l+r}{2})+f(r)}{6}$。
对于其他函数，可以同样用上面的方法，直到误差小于给定值。

Simpson 积分

对于一个二次函数 $f(x)$，有 $\int_l^r f(x)dx = (r-l)\frac{f(l)+4f(\frac{l+r}{2})+f(r)}{6}$。
对于其他函数，可以同样用上面的方法，直到误差小于给定值。
初始时可以随机划分为若干小区间。

求凸包中的最远点对

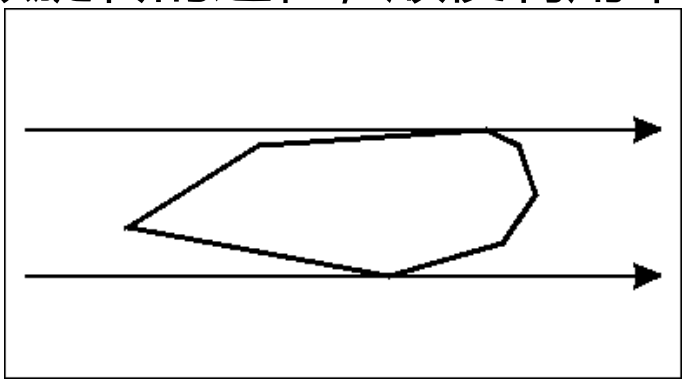
旋转卡壳算法。

求凸包中的最远点对

旋转卡壳算法。
选定两个卡壳，模拟旋转的过程，顺便利用单调性维护一些信息。

求凸包中的最远点对

旋转卡壳算法。
选定两个卡壳，模拟旋转的过程，顺便利用单调性维护一些信息。



曼哈顿距离和切比雪夫距离

在二维空间内，两个点之间的曼哈顿距离（Manhattan distance）为它们横坐标之差的绝对值与纵坐标之差的绝对值之和。设点 $A(x_1, y_1), B(x_2, y_2)$，则 A, B之间的曼哈顿距离用公式可以表示为：

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

曼哈顿距离和切比雪夫距离

在二维空间内，两个点之间的曼哈顿距离（Manhattan distance）为它们横坐标之差的绝对值与纵坐标之差的绝对值之和。设点 $A(x_1, y_1), B(x_2, y_2)$，则 A, B之间的曼哈顿距离用公式可以表示为：

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

在二维空间内，两个点之间的切比雪夫距离为它们横坐标之差的绝对值与纵坐标之差的绝对值的最大值。设点 $A(x_1, y_1), B(x_2, y_2)$，则 A, B之间的切比雪夫距离用公式可以表示为：

$$d(A, B) = max(|x_1 - x_2|, |y_1 - y_2|)$$

曼哈顿距离和切比雪夫距离

在二维空间内，两个点之间的曼哈顿距离（Manhattan distance）为它们横坐标之差的绝对值与纵坐标之差的绝对值之和。设点 $A(x_1, y_1), B(x_2, y_2)$，则 A, B之间的曼哈顿距离用公式可以表示为：

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

在二维空间内，两个点之间的切比雪夫距离为它们横坐标之差的绝对值与纵坐标之差的绝对值的最大值。设点 $A(x_1, y_1), B(x_2, y_2)$，则 A, B之间的切比雪夫距离用公式可以表示为：

$$d(A, B) = max(|x_1 - x_2|, |y_1 - y_2|)$$

所以将每一个点 (x, y)转化为 $(\frac{x+y}{2}, \frac{x-y}{2})$，新坐标系下的曼哈顿距离即为原坐标系下的切比雪夫距离。

最小圆覆盖

随机增量法。

Code Plus 某两场的题

忘了题目名

$N(N \leq 10^5)$ 个点，以两两点之间连线为直径画圆，求圆的并的周长和面积。

Code Plus 某两场的题

忘了题目名

$N(N \leq 10^5)$ 个点，以两两点之间连线为直径画圆，求圆的并的周长和面积。

拿两条夹角为 90° 的射线去旋转卡壳，初始点经过的路径就是圆并的外周。

下落的圆盘

有 $n(n \leq 10^3)$ 个圆盘从天而降，后面落下的可以盖住前面的。求最后可见轮廓线的总长。

下落的圆盘

有 $n(n \leq 10^3)$ 个圆盘从天而降，后面落下的可以盖住前面的。求最后可见轮廓线的总长。

对于每个圆盘处理出在它之后落下的圆盘在圆周上的覆盖区间。

下落的圆盘

有 $n(n \leq 10^3)$ 个圆盘从天而降，后面落下的可以盖住前面的。求最后可见轮廓线的总长。

对于每个圆盘处理出在它之后落下的圆盘在圆周上的覆盖区间。然后求一个区间并就能算出这个圆盘的可见弧长。

无题

一个点，每一时刻存在一个变换【绕原点旋转，平移，基于原点缩放】，M 组询问，每次询问假设 L_i 时刻在位置 (X_i, Y_i)，求在 R_i 时刻的位置。

无题

一个点，每一时刻存在一个变换【绕原点旋转，平移，基于原点缩放】，M 组询问，每次询问假设 L_i 时刻在位置 (X_i, Y_i)，求在 R_i 时刻的位置。

线段树 + 矩阵维护点的变化。

Disjoint Triangles

给 $n(n \leq 2000)$ 个点，求有多少对顶点是这些点的三角形不相交。

Disjoint Triangles

给 $n(n \leq 2000)$ 个点，求有多少对顶点是这些点的三角形不相交。

如果两个三角形没有交那么一定可以作出两条内公切线否则做不出来。

Disjoint Triangles

给 $n(n \leq 2000)$ 个点，求有多少对顶点是这些点的三角形不相交。

如果两个三角形没有交那么一定可以作出两条内公切线否则做不出来。
先枚举一个点，再按极角排序，枚举另一个点作为公切线，维护两边点的个数统计答案。时间复杂度 $O(n^2 \log n)$。

共点圆

有 n 次操作，每次要么加入一个圆心 (x, y) 且过原点的圆，要么询问 (x, y) 是否在所有已加入的圆的内部（含圆周）。

共点圆

有 n 次操作，每次要么加入一个圆心 (x, y) 且过原点的圆，要么询问 (x, y) 是否在所有已加入的圆的内部（含圆周）。

CDQ 分治，每次反演把圆变成直线，然后可以半平面交。

共点圆

有 n 次操作，每次要么加入一个圆心 (x, y) 且过原点的圆，要么询问 (x, y) 是否在所有已加入的圆的内部（含圆周）。

CDQ 分治，每次反演把圆变成直线，然后可以半平面交。
反演是指把一个点先转换成极坐标表示，然后把到原点的距离 d 变成 $\frac{1}{d}$。

Before Having Donuts

有 $N(N \leq 20)$ 个甜甜圈，每个甜甜圈是由参数 x, y, z, R, r 定义的一个环面，其中 (x, y, z) 为环面中心的坐标，R 为环面中心到圆管中心的半径，r 为圆管的半径。环面的旋转轴与 z 轴平行。环面可能相互接触甚至重合。请求出所有 N 的环面所占的总体积，即其并集的体积。

Before Having Donuts

有 $N(N \leq 20)$ 个甜甜圈，每个甜甜圈是由参数 x, y, z, R, r 定义的一个环面，其中 (x, y, z) 为环面中心的坐标，R 为环面中心到圆管中心的半径，r 为圆管的半径。环面的旋转轴与 z 轴平行。环面可能相互接触甚至重合。请求出所有 N 的环面所占的总体积，即其并集的体积。

考虑 Simpson 积分。每次需要对一堆圆环求并。一种方法是格林公式。

Before Having Donuts

有 $N(N \leq 20)$ 个甜甜圈，每个甜甜圈是由参数 x, y, z, R, r 定义的一个环面，其中 (x, y, z) 为环面中心的坐标，R 为环面中心到圆管中心的半径，r 为圆管的半径。环面的旋转轴与 z 轴平行。环面可能相互接触甚至重合。请求出所有 N 的环面所占的总体积，即其并集的体积。

考虑 Simpson 积分。每次需要对一堆圆环求并。一种方法是格林公式。
可以参考 https://blog.xehoth.cc/BZOJ2178/。

Before Having Donuts

有 $N(N \leq 20)$ 个甜甜圈，每个甜甜圈是由参数 x, y, z, R, r 定义的一个环面，其中 (x, y, z) 为环面中心的坐标，R 为环面中心到圆管中心的半径，r 为圆管的半径。环面的旋转轴与 z 轴平行。环面可能相互接触甚至重合。请求出所有 N 的环面所占的总体积，即其并集的体积。

考虑 Simpson 积分。每次需要对一堆圆环求并。一种方法是格林公式。
可以参考 https://blog.xehoth.cc/BZOJ2178/。
不会格林公式的话，由于这题 N 很小，可以暴力求出所有交点，然后每两个交点的横坐标中间，可以求出哪些圆弧中间有贡献。

Intranet of Buses

大小为 N 的凸多边形，M 个点在上面移动，第 i 个点初始在 $\frac{i}{M}$ 周长的位置，每个点速度相同。求最小的 D，使得存在一个时刻每对相邻点间距离都不超过 D。

$N, M \leq 10^5$。

Intranet of Buses

大小为 N 的凸多边形，M 个点在上面移动，第 i 个点初始在 $\frac{i}{M}$ 周长的位置，每个点速度相同。求最小的 D，使得存在一个时刻每对相邻点间距离都不超过 D。
$N, M \leq 10^5$。

考虑二分答案。每个点只需要移动到下一个点初始的位置，考虑这一段里面，哪些位置满足要求，然后区间求交。每一段也是在两条边上移动，可以发现和距离是二次函数关系。求出解即可。

Intranet of Buses

大小为 N 的凸多边形，M 个点在上面移动，第 i 个点初始在 $\frac{i}{M}$ 周长的位置，每个点速度相同。求最小的 D，使得存在一个时刻每对相邻点间距离都不超过 D。
$N, M \leq 10^5$。

考虑二分答案。每个点只需要移动到下一个点初始的位置，考虑这一段里面，哪些位置满足要求，然后区间求交。每一段也是在两条边上移动，可以发现和距离是二次函数关系。求出解即可。这个做法是 $O((N+M)\log V)$ 的，但实际上可以做到 $O((N+M)\log M)$。考虑对 M 个点分治，每次求出两边的函数图像，然后暴力合并。做到底部时和上面做法相同。暴力合并时，对于两段，可以求出他们相等的位置，然后进行分裂。