OI 计算几何 Part 1 基础及基本细节

Computational Geometry in OI

daklqw

Zhenhai High School

July 31, 2024

前言

- 作为退役三年的选手，现在 OI 什么版本了我也不知道。不过我准备的这些足够大家入门计算几何，剩下的需要大家自己刷题。
- Q: 题怎么找？
- A: codeforces 是最为推荐的，按标签搜。
- Q: 考得多吗？
- A: 我那几年考得不算多，但绝不是冷门，所以需要学会。
- Q: 我想要板子！
- A: 见 yhx 学长的板子 ▸ OI-transit，是我见过最好的板子。

内容一览

这两天的内容，我分成了这三部分：

- [Day 1 上午] 基础及基本细节：简单但重要。
- [Day 1 下午] 二维凸包相关：用的最多。
- [Day 2] 其他计算几何算法。

这部分太简单，我都会了

入门这部分讲大部分基础操作，较为简单。如果都会了那么可以选择：

- (理论) 课件下发了，所以可以去找课件里对应的论文都看一下。
- (推荐) 去研究一下 yhx 的板子 ▸ OI-transit，或者造一份自己的板子。
- (实用) 开始做点题，绝对不亏。
- (自学) 可以去看看 ▸ CMSC754。

OI 中的计算几何

计算几何主要研究对几何对象的算法。OI 中的计算几何有两部分重点，一部分是操作几何对象的算法，另一部分是实现。

计算几何题目是极其重实现的题目，因为其难以调试，细节较复杂。因此优秀的实现能极大提高做计算几何题的效率。

计算几何题目要求对各种基础操作熟练，并且很推荐大家做题时在纸上多画一画。

Outline

常见数据类型

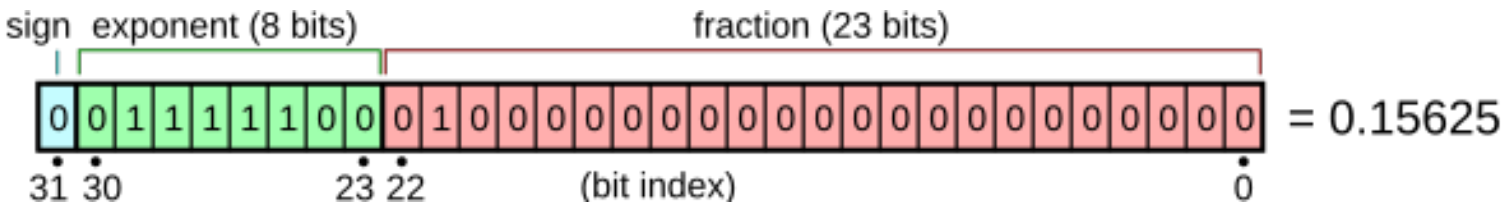
在计算几何的题目中你往往要选择合适的数据类型。

整数类型 (优先使用)：`int`, `long long`, `__int128`

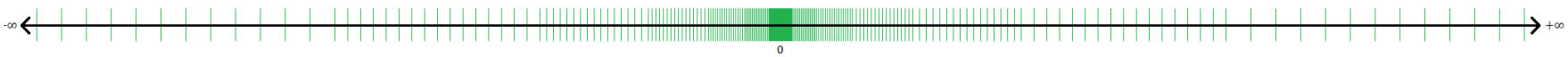
浮点类型：`float`, `double`, `long double`, `__float128` (编译环境相关，不一定能用)。

浮点数

浮点数表示为 $(-1)^{\text{sign}} \times F \times 2^E$。



可以发现，由于对于每个指数其有效数字位数是固定的。因此数轴上绝对值越接近 0，浮点数分布越密。



什么时候用整数

使用浮点数会产生各种误差，同时整数加减乘运算远快于浮点运算。因此用整数或者分数来表示几何对象是十分合适的。

分数的分母大小一般还需要细心控制，因此分数并不常用。

由于整数环和有理数环在**开根**等操作下并不封闭，所以仅适合在算法中衍生的几何对象较为简单 时使用。

一些算法，如凸包等，可以完全规避浮点数。

误差

记存下来的数为 $\hat{x}$，实际的数为 x，则：

相对误差：$\epsilon_r = \dfrac{\hat{x} - x}{x}$，绝对误差：$\epsilon = \hat{x} - x$。

一般涉及浮点数输出的题目会要求答案的相对误差或绝对误差不超过 ε，意思即为：

$$\min\left\{\left|\hat{x} - x\right|, \left|\frac{\hat{x} - x}{x}\right|\right\} \leq \varepsilon.$$

其实浮点数运算可以直接满足相对误差的限制，OI 中加上绝对误差比较无疑是更加宽松的。

浮点数

浮点数精度是有限的，会带来 $\epsilon_{\text{mach}} = 2^{-F}$ 的相对误差。

类型	位数	F 位长	$\epsilon_{\text{mach}} = 2^{-F}$
float	32	24	$\approx 6 \times 10^{-8}$
double	64	53	$\approx 1 \times 10^{-16}$
long double	80	64	$\approx 5 \times 10^{-20}$
___float128	128	113	$\approx 9 \times 10^{-35}$

OI 中常用的浮点比较

OI 中大家通常会指定一个比题目要求稍微严格的 ε，然后将数 x 看作一个区间 $[x-\varepsilon/2, x+\varepsilon/2]$：

```
const real_t eps = 1e-10;
bool less(real_t a, real_t b) { return a < b - eps; }
bool leq(real_t a, real_t b) { return a < b + eps; }
bool eq(real_t a, real_t b) { return fabs(a - b) < eps; }
```

当实际计算的误差比 ε 小的时候，这样子比较是完全正确的。

减少误差

- 可以通过避免浮点运算减少误差。
- 避免大数加减小数，例如浮点运算中 $1 + \epsilon_{\text{mach}}/2 = 1$。
- 避免大数减去相近的大数。
- 使用误差较小的算法，例如选主元的高斯消元。

深入学习

如果想对数值相关问题有更深入的了解，可以学习数值分析 (Numerical analysis)。

本小节涉及到的浮点数材料：

- IEEE 754 浮点运算标准 - Wikipedia Link
- 扩展精度格式 - Wikipedia Link

Outline

点 (Point)

一般来说，采用笛卡尔座标系 (Cartesian coordinate system) 来记录点的位置。并且出题人也会这么出。

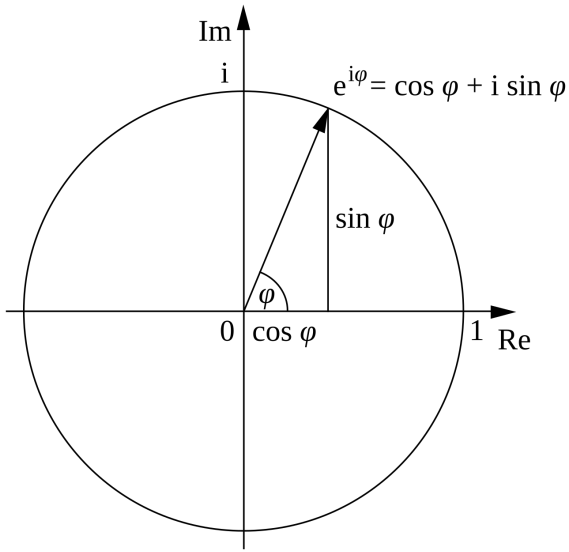
```
struct Point2D {
  int x, y;
};

struct Point3D {
  int x, y, z;
};
```

点的极座标表示

对于点 $(r\cos\theta, r\sin\theta)$，可以表示为极座标 (r, θ)。

对应着复平面 (Complex plane) 中的点 $r\exp(i\theta) = r\cos\theta + ir\sin\theta$。

这种表现形式不常用，但是某些题目中会很好用。



Example (区间求 sin)

你需要维护一个序列 $A = (a_1, a_2, \ldots, a_n)$，支持：

1. 将区间 $[l, r]$ 内的 a_i 加上值 x（区间加），
2. 对于区间 $[l, r]$ 求 $\sum_{i=l}^{r} \sin a_i$（区间求 sin）。

数据范围 $n \leq 10^6$，操作数 $\leq 10^6$。

Example (区间求 sin)

你需要维护一个序列 $A = (a_1, a_2, \ldots, a_n)$，支持：

1. 将区间 $[l, r]$ 内的 a_i 加上值 x（区间加），
2. 对于区间 $[l, r]$ 求 $\sum_{i=l}^{r} \sin a_i$（区间求 sin）。

数据范围 $n \le 10^6$，操作数 $\le 10^6$。

比较经典的区间操作，注意到 $\sin x = \mathrm{Im} e^{ix}$，并且 $\mathrm{Im}(a+b) = \mathrm{Im}a + \mathrm{Im}b$。因此区间 $\sin$ 和对应着区间复数 $\exp(ia_j)$ 的和。

对于区间加，相当于区间里的 $\exp(ia_j) \to \exp(i(a_j + x)) = \exp(ia_j)\exp(ix)$，相当于一个区间乘法。

区间求 sin

其实上面一道例题使用 $\sin(a+b)=\sin a\cos b+\sin b\cos a$，使用矩阵乘法也可以解决。但是本质上都是使用复数。

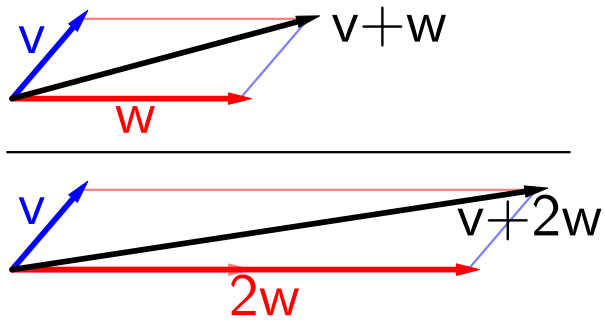
这么讲述极坐标有点牵强。实际上极坐标在某些情况里确实有更好的表现，例如某些涉及绕点旋转的计算。

但是极坐标在计算上也需要一些修改，这涉及到微积分。比如在圆轮廓长度的微分 $\mathrm{d}\ell=r\mathrm{d}\theta$。

向量 (Vector)

可以看作是从坐标原点到点 P 的向量，因此可以用点相同的方法存储。

向量需要支持加减法，倍乘等操作。



内积 (Inner Product)

在欧几里得空间中，内积 $\langle a, b\rangle$ 定义为 $\sum_{i=1}^{n} a_i b_i$。

其满足定义内积要求的几条性质：

- $\langle x, y\rangle = \langle y, x\rangle$,
- $\langle ax + by, z\rangle = a\langle x, z\rangle + b\langle y, z\rangle$,
- $\langle x, x\rangle \geq 0$，等号成立当且仅当 $x = 0$。

通过内积可以定义向量的模长，即二范数：

$$\|x\|_2 = |x| = \sqrt{\langle x, x\rangle} = \sqrt{\sum_i x_i^2}$$

Proposition

对于任意向量 x, y，$\langle x, y\rangle = |x||y|\cos\theta$，其中 θ 是向量 x, y 的夹角。

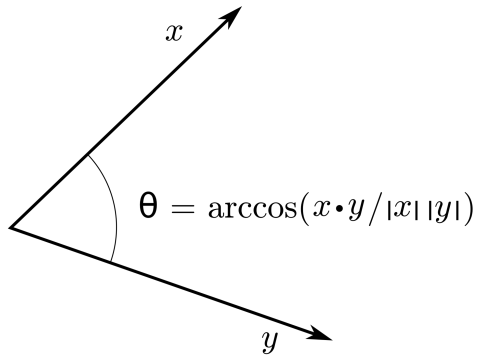
Proposition

对于任意向量 x, y，$\langle x, y\rangle = |x||y|\cos\theta$，其中 θ 是向量 x, y 的夹角。

Proof.

考虑 $y = y_\perp + \alpha x$，其中 $\langle x, y_\perp\rangle = 0, \alpha = \langle x, y\rangle$。

此时 αx 对应着 y 到 x 上的投影，即 $|y|\cos\theta$。 □



单纯形 (Simplex)

Definition (Simplex)

n 维单纯形为 $n+1$ 个顶点构成的凸包，即

$$C = \{\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_{n+1} x_{n+1} : \alpha_1 + \alpha_2 + \ldots \alpha_{n+1} = 1, \alpha_i \geq 0\}.$$

单纯形的体积

Proposition

选取 n 维单纯形 C 任意一个顶点 u_0，令 u_0 到其他 n 个点 u_i 的向量为 $x_i = u_i - u_0$，则单纯形 C 的体积为

$$\mathrm{vol}(C) = \frac{1}{n!} \left|\det(x_1, x_2, \ldots, x_n)\right|$$

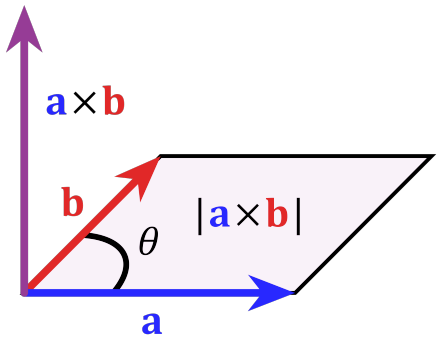
即对应行列式的绝对值除以 $n!$。

单纯形的体积

Example (三角形面积)

$\frac{1}{2}|\det(x,y)| = \frac{1}{2}|x||y|\,|\sin\theta|$，提供了一种计算两向量夹角正弦值的方法。

注意到行列式 $\det$ 实际上是反对称的，即 $\det(x,y) = \det(y,x)$。一般约定逆时针方向为正方向，即 θ 满足 $\det(x,y) = |x||y|\sin\theta$。此时 $\det(x,y) = x \times y$ 又称为叉积 (Cross Product)。



求向量夹角

Proposition

给定向量 x, y, 则向量之间夹角 θ 满足

$$\tan\theta = \frac{\sin\theta}{\cos\theta} = \frac{x \times y}{\langle x, y\rangle}$$

因此 $\theta = \arctan \dfrac{x \times y}{\langle x, y\rangle}$.

由于 $\arctan$ 是单调函数，因此可以直接通过比较 $\dfrac{x \times y}{\langle x, y\rangle}$ 来比较夹角之间大小。

求向量所在象限

Example

给定向量 x, y，求 y 在 x 逆时针方向的象限。

即令向量 x 为 x 轴正方向，逆时针方向为正方向，求 y 在第几象限。

观察 $\cos\theta$ 和 $\sin\theta$ 的正负性，即 $\langle x, y\rangle$ 和 $x \times y$ 的正负性，得到：

象限	$\langle x, y\rangle$	$x \times y$
第一象限	> 0	> 0
第二象限	< 0	> 0
第三象限	< 0	< 0
第四象限	> 0	< 0

向量平行与垂直

Example

给定向量 x, y，问 x, y 是否平行，是否垂直。

x, y 平行当且仅当 $x \times y = 0$,

x, y 垂直当且仅当 $\langle x, y \rangle = 0$.

三点共线

Example

给定三个二维平面点 a, b, c，问是否三点共线。

可以直接使用 $b-a$ 和 $c-a$ 的叉积判断。

另一种方法就是计算行列式：

$$\det \begin{pmatrix} x_a & x_b & x_c \\ y_a & y_b & y_c \\ 1 & 1 & 1 \end{pmatrix}$$

即将其变为三维空间中 $z=1$ 的点，此时单纯形体积为 0 当且仅当三点共线。

极角排序

Example

给定 n 个向量 a_i，令向量 x 为 x 轴正方向，将所有向量以 x 到 a_i 夹角的大小进行排序。要求 $O(n \log n)$ 算法。

这样排序得到的序列为一个环状结构，即排序后的向量中，取任意一个向量作为 x 轴正方向，从这个向量开始的序列夹角都是不降的。

```
bool less(vec2d a, vec2d b) {
  if (a.quadrant() == b.quadrant()) {
    int cross = cross_product(a, b);
    if (cross == 0) return a.norm() < b.norm();
    return corss > 0;
  }
  return a.quadrant() < b.quadrant();
}
```

极角排序

一般来说，问题会给一堆点，选取一个源点，得到 $n-1$ 个从这个源点出发的向量，进行极角排序。如果这个源点是任意的，只需要选择一个凸包上的点作为源点，即可避免大量细节：

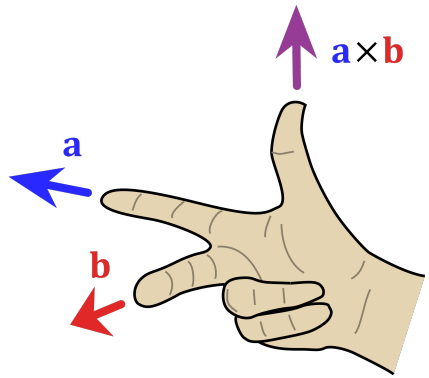
```
bool less(vec2d a, vec2d b) {
  int cross = cross_product(a, b);
  if (cross == 0) return a.norm() < b.norm();
  return corss > 0;
}
```

这需要只存在两个象限的向量，因此需要以 x 为第一关键字，y 为第二关键字，取最小的点（第二关键字十分重要，不然要额外代码来区分类似 $(-1,0)$ 和 $(1,0)$ 的向量）

三维向量叉积 (Cross Product)

Definition

给定两个三维向量 x, y，向量叉积定义为 $x \times y = |x||y| \sin\theta \cdot n$ 其中 n 为满足右手定则的单位向量。使用行列式的形式表示为



$$a \times b = \det \begin{pmatrix} x_a & x_b & \mathbf{i} \\ y_a & y_b & \mathbf{j} \\ z_a & z_b & \mathbf{k} \end{pmatrix},$$

其中 i, j, k 为 x, y, z 方向的单位向量。

可以用来计算与 x, y 平面正交的向量 z。

三维向量混合积 (Triple product)

Definition

给定三维向量 a, b, c，它们的混合积为

$$\det(a, b, c) = \langle a, b \times c \rangle = \langle b, c \times a \rangle = \langle c, a \times b \rangle$$

直线 (1)

直线有多种表示形式：

1. 两点确定一条直线：记录两个点 x, y。
2. 直线过一点：写成 $x + kv$ 的形式。
3. 直线方程：$Ax + By + C = 0$，(A, B) 为直线的法向量。

直线方程：投影到 (A, B) 的单位向量上长度为 $-C$ 的点。

三种形式可以互相转换：

- (2) $\Rightarrow$ (1)：令 $k = 0, k = 1$。
- (3) $\Rightarrow$ (1)：令 $x = 0, x = B$，分别解方程。

直线 (2)

1. 两点确定一条直线：记录两个点 x, y。
2. 直线过一点：写成 $x + kv$ 的形式。
3. 直线方程：$Ax + By + C = 0$。
   - (1) $\Rightarrow$ (2)：令 $v = y - x$。
   - (3) $\Rightarrow$ (2)：令 $x = 0$ 解出基点，$v = (B, -A)$。

直线 (3)

1. 两点确定一条直线：记录两个点 x, y。
2. 直线过一点：写成 $x + kv$ 的形式。
3. 直线方程：$Ax + By + C = 0$。
   - (1) $\Rightarrow$ (3)：令 $C = x \times y, (B, -A) = y - x$。
   - (2) $\Rightarrow$ (3)：令 $C = x \times v, (B, -A) = v$。

直线交点

Example

给定两条不平行的直线，求它们的交点。

Algorithm

相当于解二元一次方程。例如两个直线方程，则需要解

$$\begin{cases} A_1 x + B_1 y + C_1 = 0 \\ A_2 x + B_2 y + C_2 = 0 \end{cases}$$

直线垂直/平行

Example

判断直线 l_1, l_2 是否平行，是否垂直。

由于法向量和直线是垂直的，因此根据直线的形式可以得到其法向量，通过比较法向量之间关系可以得到直线之间关系。

直线交点

Algorithm

如果一个是直线过一点的形式，另一个是直线方程，则形式为：

$$\langle (A,B), x+kv \rangle + C = 0$$

根据内积的线性性质，可以得到一个一元一次方程，解得

$$k = -\frac{C + \langle (A,B), x \rangle}{\langle (A,B), v \rangle}.$$

计算准则

Principle

一般来说，几何对象的表现形式有两种，一种是方程形式，另一种是参数形式。

例如 $Ax+By+C=0$ 是方程形式，而 $x+kv$ 是参数形式。

一般来说，两个对象求交时，两者使用不同表现形式会简化计算过程，即一个为方程形式，另一个为参数形式。

不过一般来说，为了代码写的方便这种情况还是用的不多。

点到直线投影/反射

Example

求点 p 到直线 l 的投影 q 与反射 p'。
反射 p' 满足 $(p+p')/2=q$。

Algorithm

即求过点 p，方向为直线 l 的法向量 n 的直线与 l 的交点。
由于直线的几何意义是：到其法向量投影长度固定的点集，所以通过点和法向量的内积算出点到法向量投影长度。
然后可以算出参数 k（两个长度的差）而得到交点 $p+kn$。

中垂线

Example

给定两点 u, v，求 u, v 的中垂线。

Algorithm

中垂线的法方向为 $v-u$。考虑 u 和 v 到中垂线的距离相反，所以

$$\begin{cases} \langle v-u, u\rangle + C = k \\ \langle v-u, v\rangle + C = -k \end{cases}$$

$$\Rightarrow C = \frac{1}{2}\langle u-v, v+u\rangle = \frac{1}{2}(|u|^2 - |v|^2)$$

角平分线

Example

给定两条直线，求他们之间的角平分线。

Algorithm

相当于取法向量的角平分线。为了简化计算，我们需要两个单位化的法向量 u, v，即 $|u| = |v| = 1$。

此时连接 u, v，线段和角平分线垂直。因此通过 $v - u$ 就可以算出直线的方向。通过计算两直线的交点可以得到 C。

角平分线

```
line bisector(const line &l1, const line &l2) {
  vec2d u = l1.norm(), v = l2.norm();
  vec2d dir = v.normalize() - u.normalize();
  vec2d w = intersection(l1, l2);
  real_t C = inner_product(dir, w);
  return {dir.x, dir.y, -C};
}
```

交换两条直线即可得到另一条角平分线。

射线与线段

射线和线段是直线的一部分。

对于射线，最好使用 $u + kv$ 的参数形式表示。

对于线段，最好使用两点 u, v 的形式表示。

点在线段/射线上

Example

给定线段 u, v，求点 p 是否在线段上。

给定射线 $u + kv$，求点 p 是否在射线上。

点在线段上需要计算 $p - u$ 和 $p - v$ 是否反向（内积 ≤ 0，叉积 $= 0$）

点在射线上要判断 $p - u$ 是否和 v 同向（内积 ≥ 0，叉积 $= 0$）

直线操作拓展到这两种线上一般只需要判断是否点在线段上。

线段相交

Example

给定线段 u_1, v_1，和线段 u_2, v_2，求两个线段是否相交。

求交点可能会带来精度问题。实际上只需要判断线段的两端点是否在另一条线段的两侧即可。使用叉积获得角的方向即可快速判断：

```
bool across(segment a, segment b) {
  int x = cross(a.u - b.u, b.v - b.u);
  int y = cross(a.v - b.u, b.v - b.u);
  return (x <= 0 && y >= 0) || (x >= 0 && y <= 0);
}
bool intersect(segment a, segment b) {
  return across(a, b) && across(b, a);
}
```

Outline

线性映射

Definition (Linear Map)

对于线性空间 V, W，映射 $f: V \mapsto W$ 是一个线性映射当：

- $\forall x, y \in V, f(x+y) = f(x) + f(y)$,
- $\forall x \in V, a \in \mathbb{F}, f(ax) = af(x)$.

Proposition

如果 V, W 是有限维的，那么对于 V, W 的任意一组基 $\{v_1, \ldots, v_n\}$ 以及 $\{w_1, \ldots, w_m\}$，f 可以表示为矩阵 $[f(v_1), f(v_2), \ldots, f(v_n)]$。
其中第 j 行第 i 列是 $f(v_i)$ 在 w_j 上的分量。

Proposition

如果 V, W 是有限维的，那么对于 V, W 的任意一组基 $\{v_1, \ldots, v_n\}$ 以及 $\{w_1, \ldots, w_m\}$，矩阵 $[f(v_1), f(v_2), \ldots, f(v_n)]$ 唯一确定线性映射 f。

线性映射的复合

Proposition

对于线性映射 $f: U \mapsto V, g: V \mapsto W$，**对应矩阵** A, B，**那么** $g \circ f: U \mapsto W$ **在与** f, g **相同的基下可以表示为** $C = BA$，**即矩阵乘法**：

$$c_{i,j} = \sum_{k=1}^{|V|} b_{i,k} a_{k,j}.$$

几何对象常见线性映射

拉伸

$$\begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix}$$

旋转

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

反射（对向量 w）

$$I - 2ww^T$$

平移

平移在二维矩阵下并不是线性的，为此，我们需要引入新的一维表示常数。

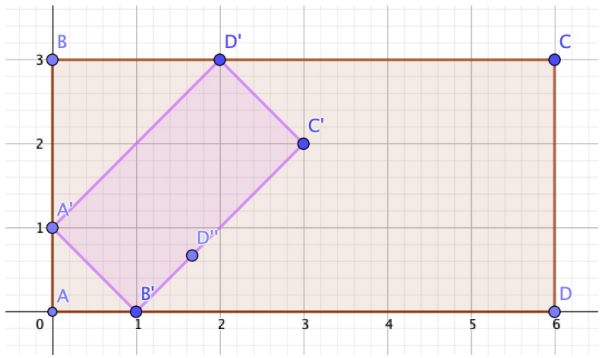
所有向量 (a, b) 变成 $(a, b, 1)$，此时平移矩阵 $x + x_0, y + y_0$ 表示为：

$$\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Example (EC-Final 2022 E Map ▸ vjudge)

你在一个矩形的场地 F 内，有一张矩形的地图 M 放在这个矩形的场地内部。这个矩形地图 M 比例对应这个场地，即存在双射 $f: F \mapsto M$ 以及常数 $r > 1$，使得：$\forall a, b \in F, (|f(a) - f(b)|)/(|a - b|) = r.$

你每单位时间可以走单位距离，你的任务是从 S 点走到 T 点。当你在矩形上的时候，你可以选择传送到地图上对应的位置，即令 $x' = f(x)$。当你在地图上的时候，你可以选择传送到矩形上对应的位置，即令 $x' = f^{-1}(x)$。每次传送要消耗 t 的时间，你有 n 次传送机会。求从 S 点到达 T 点要花费的最少时间。



数据组数 $T \leq 100$，$n \leq 100$。

EC-Final 2022 E Map

Solution

注意到矩形到地图的映射实际上是一个线性映射。令 W 表示行走，此时行动的序列为 $(W_1, f_1, W_2, f_2, W_3, \ldots, W_n, f_n, W_{n+1})$，其中 f_i 可以是 f 或 f'。

注意到，如果 $f_i = f(x)$，则 $W, f = f, f(W)$，其中 $f(W)$ 行走的距离比 W 少，因为地图比场地更小。

同理如果 $f_i = f^{-1}(x)$，则 $f^{-1}, W = f^{-1}(W), f^{-1}$，交换后行走距离同样变短。

所以最优做法是直接通过 f 从 S 传送若干次，然后走一段距离，再通过 f^{-1} 传送若干次到 T。因此枚举 f 和 f^{-1} 的次数即可。

EC-Final 2022 E Map

Solution (Cont.)

那么，如何表示出这个映射呢？注意到任意线性映射可以使用一组基及其对应的像来表示，分别在矩形和地图的矩形边上取一组正交基，则其中任意的点都有一个坐标 (x, y)。这个坐标可以通过与矩形边做内积来获得。

对于映射 $f(a, b)$，在矩形内找到地图基 $\hat{x}, \hat{y}$ 对应的坐标 $\tilde{x}, \tilde{y}$，那么 $f(a, b) = a\tilde{x} + b\tilde{y}$。

对于映射 $f^{-1}(a, b)$，同样找到矩形的基对应的坐标即可表示出变换。

时间复杂度 $O(Tn^2)$。

Outline

圆

圆有两种表示方法：

- 中心为 x，半径为 r 的圆。可以表示为参数形式 $x+(r\cos\theta, r\sin\theta)$。
- 圆方程：$(x-a)^2+(y-b)^2=r^2$。

点与圆关系

Example

给定圆和点，问点是在圆内，圆外还是圆上。

直接通过点到圆心的距离判断。

直线与圆交点

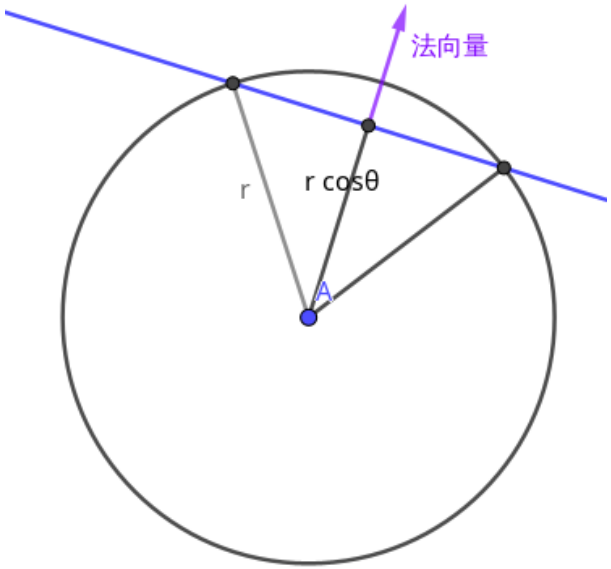
Example

给定圆和直线，问直线和圆的交点在什么位置。

Algorithm

通过圆心到直线距离即可知道交点个数。一般情况是有两个交点，我们通过旋转直线的法向量，来得到圆心到两个交点的方向。

此时由于圆心到直线距离 $r\cos\theta$ 已知，圆半径 r 已知，因此可以计算出 $\sin\theta, \cos\theta$ 得到旋转矩阵。



圆与圆关系

Example

给定两圆，问圆是相交，相离，还是包含。

Algorithm

假设 $r_1 \leq r_2$。

相离：圆心距离 $d > (r_1 + r_2)$。

包含：圆心距离 $d < (r_2 - r_1)$。

相交：$r_2 - r_1 \leq d \leq r_1 + r_2$。

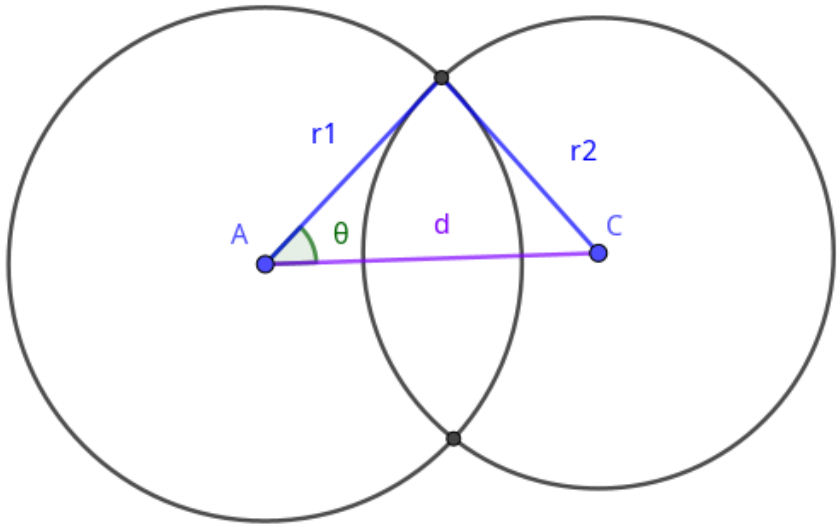
圆与圆交点

Example

给定相交的两圆，问圆的交点。

Algorithm

和直线与圆交点类似，使用余弦定理可以计算出 $\cos\theta = \dfrac{r_1^2 + d^2 - r_2^2}{2r_1 d}$。通过旋转矩阵可以得到交点。



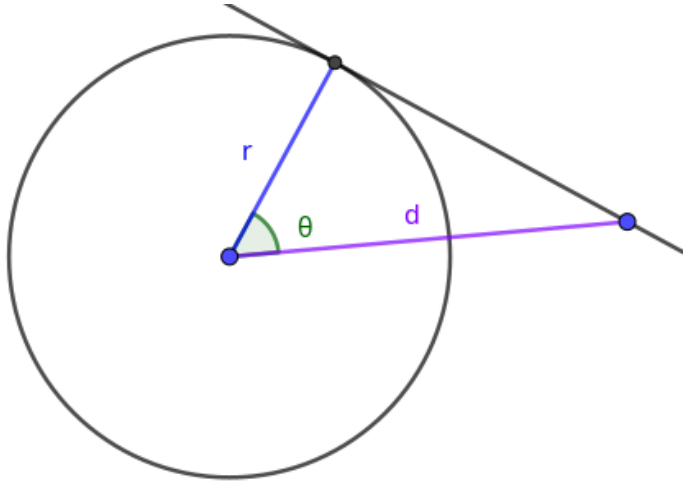
三种切线

Example (切线)

给定圆和点，求点到圆的切线。

Algorithm

对于切线，可以得到 $\cos\theta = \frac{r}{d}$.

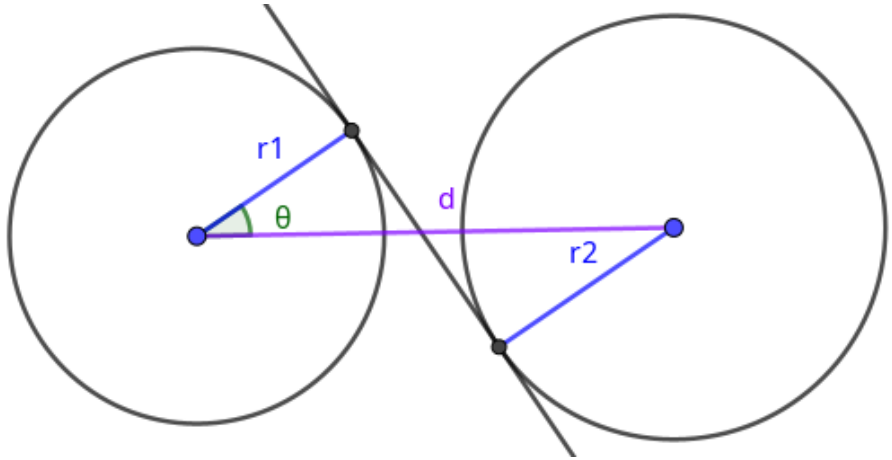


Example (内公切线)

给定分离的两圆，求内公切线。

Algorithm

对于内公切线，可以得到 $\cos\theta = \dfrac{r_1 + r_2}{d}$.

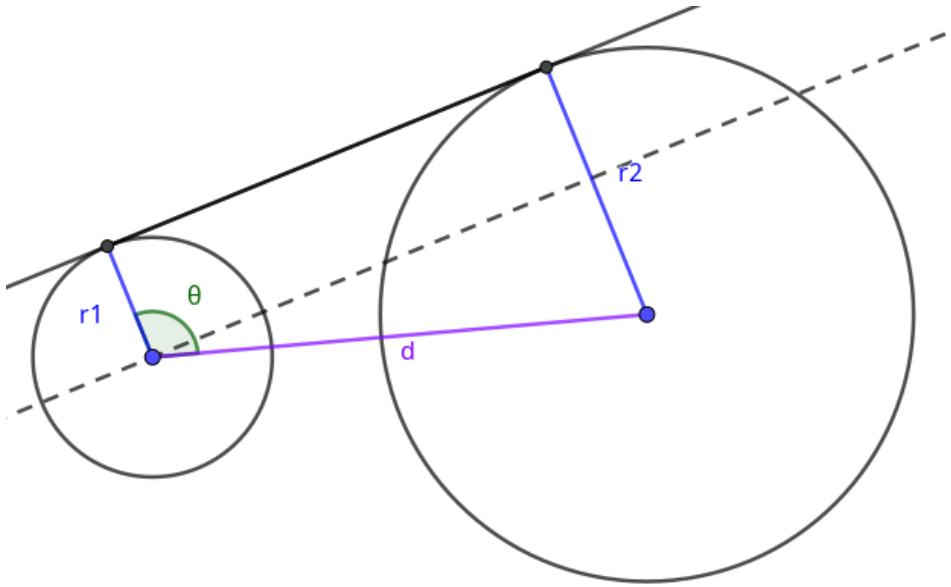


Example (内公切线)

给定不包含的两圆，求外公切线。

Algorithm

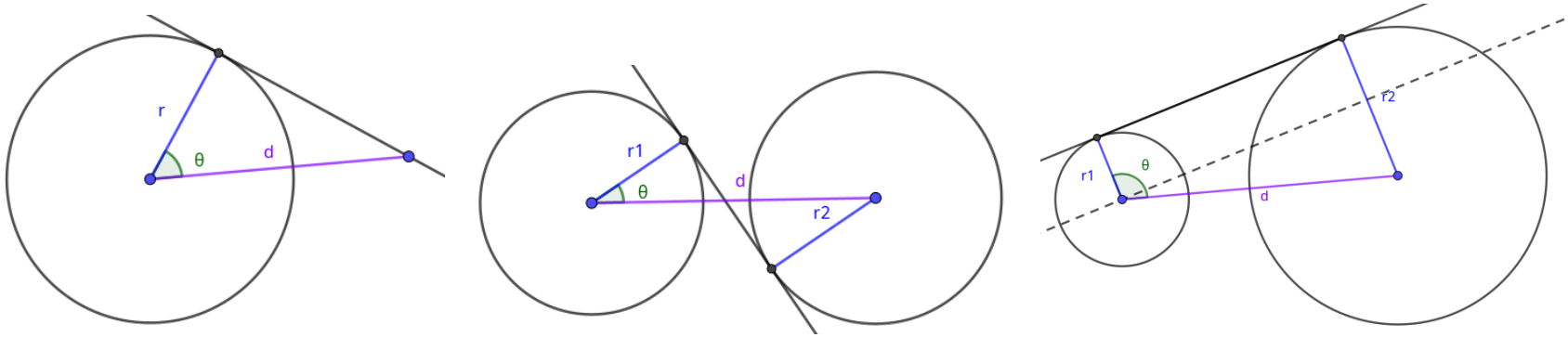
对于外公切线，可以得到 $\cos\theta = \dfrac{r_2 - r_1}{d}$.



三种切线

Algorithm

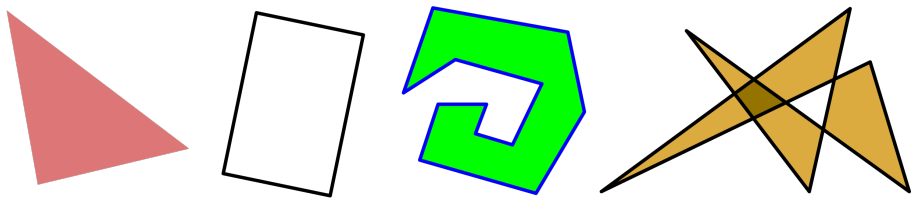
已知向量 d，通过旋转可以得到切点。其实旋转后直接就是切线的法向量，因此通过代入圆心坐标即可解出直线方程中的 C。



多边形

多边形一般以首尾相接的点序列存储，也就是点的数组 $a_1 \ldots, a_n$。表示 $(a_1, a_2), (a_3, a_4), \ldots, (a_{n-1}, a_n), (a_n, a_1)$ 是多边形的 n 条边。

一般来说，题目中遇到的都是简单多边形，即边没有相交的多边形。



凸包

Definition (Convex Hull)

给定 n 个点，其构成的凸包为

$$C = \{\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n : \alpha_1 + \alpha_2 + \ldots \alpha_n = 1, \alpha_i \geq 0\}.$$

也就是包含点集的最小凸多边形。一般按逆时针顺序存下凸包边界上的顶点。

凸，即任意相邻三点中，第三个点一定在前两个点构成线段的左侧。形式化地讲，就是 $(p_{i+1} - p_i) \times (p_{i+2} - p_i) > 0$。

关于凸性和凸包的算法，将会在之后讲述。

Outline

3SUM 问题

Example (3SUM Problem)

给定 n 个整数，问是否存在三个整数 a, b, c 使得 $a + b + c = 0$。整数的值域没有限制。

3SUM 问题

Example (3SUM Problem)

给定 n 个整数，问是否存在三个整数 a, b, c 使得 $a + b + c = 0$。整数的值域没有限制。

Algorithm (3SUM Problem)

将所有数排序，枚举 c，则有 $a + b = -c$。用双指针从小到大枚举 a，寻找对应的 b。

时间复杂度 $O(n^2)$。

3SUM-Hardness

Open Problem

3SUM 问题是否能在 $O(n^{2-\varepsilon})$ 时间内解决？其中 $\varepsilon > 0$。

3SUM-Conjecture [A Abboud, VV Williams, O Weimann, 2014]

在字长为 $O(\log n)$ 的 Word RAM 模型中，对于权值在 $[-n^3, n^3]$ 的 3SUM 问题需要 $n^{2-o(1)}$ 时间解决。

Definition (3SUM-Hardness)

一个问题是 3SUM-Hard 的当：如果这个问题能在 $O(n^{2-\delta_1})$ 时间内解决，那么 $\exists \delta_2$，3SUM 问题能在 $O(n^{2-\delta_2})$ 时间内解决。

Example (三点共线)

给定 n 个点，判断其中有没有三个点共线。要求 $O(n^2)$。

Example (三点共线)

给定 n 个点，判断其中有没有三个点共线。要求 $O(n^2)$。

Algorithm ($O(n^2 \log n)$)

枚举三个点中的任意一点，将剩下的点按照极角排序，然后判断是否有两个向量共线。

Example (三点共线)

给定 n 个点，判断其中有没有三个点共线。要求 $O(n^2)$。

Algorithm ($O(n^2 \log n)$)

枚举三个点中的任意一点，将剩下的点按照极角排序，然后判断是否有两个向量共线。

Algorithm ($O(n^2)$)

同样枚举任意一点，剩下的向量判重通过哈希表。几个思路：

- 将斜率扔进 *Hash* 表。
- 构造一个大的 *Bounding Box*，使得所有点在 *Box* 里面，把向量与 *Box* 的交点扔进 *Hash* 表。

Proposition

三点共线是 *3SUM-Hard* 的。如果 *3SUM-Conjecture* 成立则不存在低于二次的算法。

Proof.

对于 3SUM 问题，将 n 个不同的数 a_i 映射到点 (a_i, a_i^3)。考虑计算

$$\det \begin{pmatrix} 1 & 1 & 1 \\ a & b & c \\ a^3 & b^3 & c^3 \end{pmatrix} = (a-b)(b-c)(c-a)(a+b+c)$$

由于 a_i 两两不同，此时三点共线当且仅当 $a+b+c=0$。 □

Example (三线共点)

给定 n 条直线，判断其中有没有三条直线共点。要求 $O(n^2)$。

Algorithm ($O(n^2)$)

直接将两两直线的交点扔进 *Hash* 表，判断是否有重复。

Proposition

三线共点是 *3SUM-Hard* 的。如果 *3SUM-Conjecture* 成立则不存在低于二次的算法。

点线对偶.

若点 (a, b) 在 $Ax + By - 1 = 0$ 上，则直线 $ax + by - 1 = 0$ 过点 (A, B).

因此三点共线对应着三线共点。 □

其他 3SUM-Hard 问题

具体见论文 On a class of $O(n^2)$ problems in computational geometry [A Gajentaan, MH Overmars, 1995]。

- 给定若干垂直线段，是否存在一条直线将它们分成两部分
- 给定若干纸带（两平行线中间面积），问它们的并是否包含某矩形
- 给定若干三角形，问它们的并是否包含某三角形
- 给定若干三角形，问它们的并是否有洞
- 三角形面积并
- ……