0.声明

1.点的存储

点的保存为直角坐标系，如下

```
class Point{
public:
    ld x , y; //ld = long double
};
```

2.边的存储

边的保存为两个点(p , v)，其中p为直线上某一点，v为方向向量，$angle$是直线的倾角（用于排序）

```
class Line{
public:
    Point p , v;
    ld angle;
};
```

3.函数使用

求两直线交点（调用前需保证两直线不能平行）

```
Point Linewithline(Line a , Line b){
    Point u = a.p - b.p;
    ld t = (b.v * u) / (a.v * b.v);
    return a.p + a.v * t;
}
```

判断一个点是否在某一条直线对应的半平面内（严格内部）

```
bool BothSide(Point p , Line a){
    return a.v * (p - a.p) > eps;
}
```

1.引入

在一个无限大的平面上，有n条直线。每条直线相当于一次切割，只保留目前平面中这条直线左边或者右边的部分，求最终剩余平面。

由于每条直线的切割可以看成一个半平面，最终平面可以看成这n个半平面的交。

2.算法介绍

1.朴素的做法

此做法不做为讲述重点。

我们可以存储半平面对应的所有交点和交线，对于每一个新的半平面，我们暴力的枚举每一条线，并把这条线和半平面对应的分界线求交点。

当合法的交点数恰好为2时，我们可以根据方向留下那一半的直线和点，删去另一半直线和点，加入这条直线和两个新的交点。

否则对应这个半平面我们不做任何处理。

该方法时间复杂度为$O(n^2)$

此方法由于复杂度较高，在大部分题目中都无法使用，因此在这里不提供代码了。

2.优化的做法

在朴素的做法中，我们对于最终的半平面交是一个凸包的这个性质并没有使用。如果我们有效的利用这个性质，将能得到时间复杂度更低的做法。

首先我们按照半平面的分界线的方向角度$angle$排序，我们尝试去维护一个下凸壳（这个下凸壳包围的区域就是目前得到的半平面）。

```
d
d
```