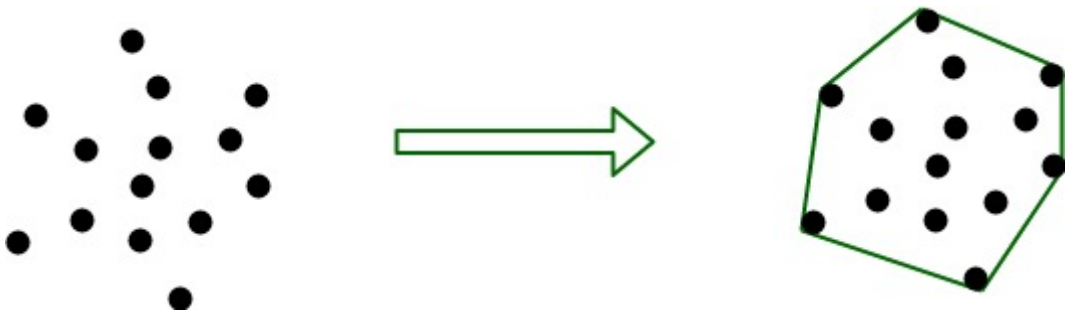
凸包

凸包（Convex Hull）是一个计算几何（图形学）中的概念。点集Q的凸包是指一个最小凸多边形，满足Q中的点或者在多边形边上或者在其内。

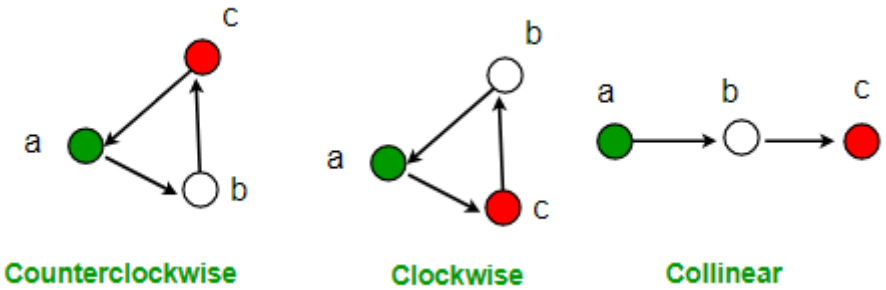


正式讨论凸包问题之前，这里先引入一个辅助概念——“方向”。

有序点的方向

一个平面内有序点的方向（Orientation）可以有三种：

- 逆时针 CounterClockwise
- 顺时针 Clockwise
- 共线 Colinear



对于点 $a(x_1, y_1)$、$b(x_2, y_2)$、$c(x_3, y_3)$，

线段ab的斜率为

$$\sigma = \frac{y_2 - y_1}{x_2 - x_1}$$

线段bc的斜率为

$$\tau = \frac{y_3 - y_2}{x_3 - x_2}$$

- 若$\sigma < \tau$，方向是逆时针（向左转）
- 若$\sigma = \tau$，方向是共线
- 若$\sigma > \tau$，方向是顺时针（向右转）

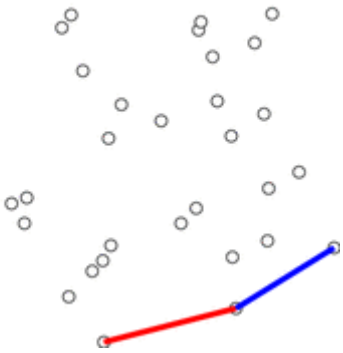
因此，三个有序点的方向依赖于表达式（通分得到）

$$(y_2 - y_1) \times (x_3 - x_2) - (y_3 - y_2) \times (x_2 - x_1)$$

- 若表达式为负，方向是逆时针
- 若表达式为0，方向是共线
- 若表达式为正，方向是顺时针

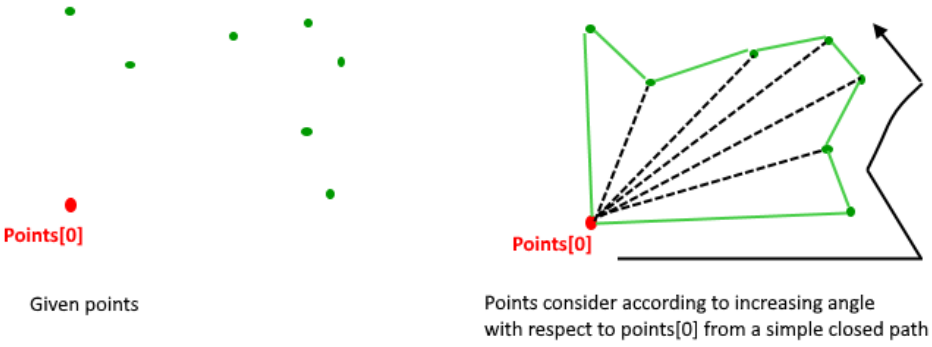
Graham Scan

```
let points be the list of points
let stack = empty_stack()

find the lowest y-coordinate and leftmost point, called P0
sort points by polar angle with P0, if several points have the same polar angle then only keep the farthest

for point in points:
    # pop the last point from the stack if we turn clockwise to reach this point
    while count stack > 1 and ccw(next_to_top(stack), top(stack), point) <= 0:
        pop stack
    push point to stack
end
```



算法可以分为两个主要部分：

1. 预处理
    1. 找到最左下方的点。使该点 p0 作为输出凸包的第一个元素 points[0]。
    2. 将剩下的 n - 1 个点按照与p0的极角序排序，若有角度相同，仅保留距离 p0 最远的那个点



Given points

Points consider according to increasing angle
with respect to points[0] from a simple closed path

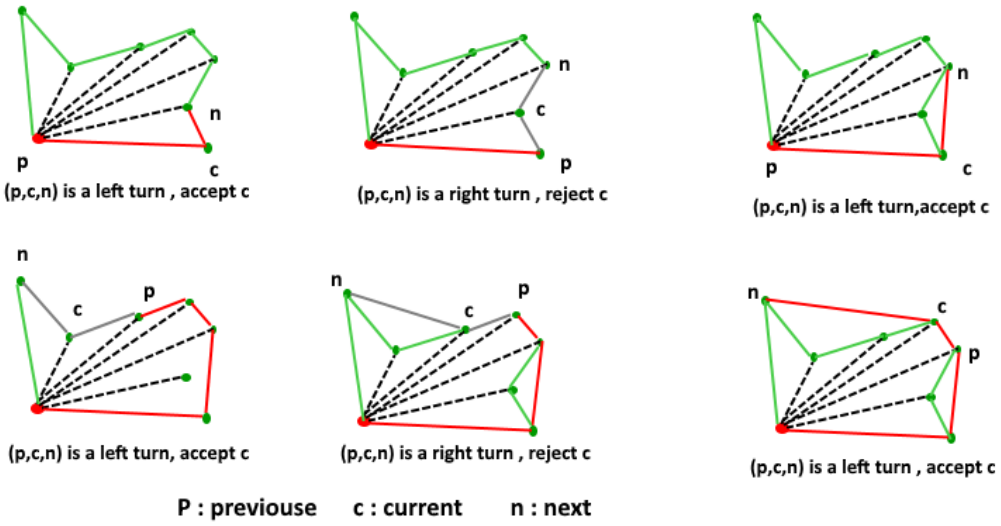
2. 接受或拒绝点

    1. 创建空栈 S，将 【栈顶的下一个点、位于栈顶的点 】 入栈。
    2. 处理剩余的每个 points[i]：
    3. 追踪当前的三个点：栈顶的下一个点、位于栈顶的点，当前分析的点points[i]。三点之间有两条连线，看作是两个向量，计算他们之间的叉积，返回三点之间的关系：

        <0，说明第三个点是向左转，则保留第二个点（栈顶元素），将第三个点进栈

        >0，说明第三个点是向右转，则删除第二个点（栈顶元素），再将第三个点进栈

        =0，说明三点共线（可采用>0的处理方式）



In the above algorithm and below code , a stack of points is used to store convex hull points. With reference to the code ,p is next-to-top in stack, c is top of stack and n is point[i]

算法的第 1.1 步（找到最左下方的点）花 O(n) 时间，第 1.2 步（点的排序）花 O(n * logn) 时间。

第 2 个步骤中，每个元素入栈和出栈最多一次，假设栈操作 O(1) 时间，则第 2 步总共花 O(n) 时间。因此总体的时间复杂度是 O(n * logn)。

代码

```python
import math
points = []



def read_file(file):
    points = []
    with open(file, 'r') as f:
        for line in f:
            key, x, y = line.strip('\n').split(',')
            points.append([int(key), float(x), float(y)])   # 每行的代表一个点，三个值分别为：编号、横坐标、纵坐标
    return points




# 分别求出后面n-1个点与出发点的斜率，借助sorted完成从小到大排序
def compute(next):
    start = points[0]  # 第一个点

    # 按极角序排列的方法，但现在输入的坐标点是笛卡尔坐标点。不适合用这个
    # angle = math.atan2( start[2] - next[2], start[1] - next[1] )
    # return angle


    # 按斜率排列的方法
    if start[1] == next[1]:  # 如果x坐标相同，那么求斜率时会出现分母为0的情况,直接返回斜率无穷大
        return 99999
    slope = (start[2] - next[2]) / (start[1] - next[1])
    return slope









def Graham_Scan(points):

    # points.sort(key=lambda x:x[1])   # 按横坐标排序的方法（和维基伪代码思想不太一致，弃用）

    # 找到最左边且最下面的点作为出发点，和第一位互换
    Min=9999
    for i in points:
        # 寻找最下边的点
        if i[1]<Min:
            Min = i[1]
            index = i[0]-1
        # 如果同在最左边，可取y值更小的
```

```
        elif i[1]==Min:
            if i[2]<=points[index][2]:
                Min = i[1]
                index = i[0]-1
    # 和第一位互换位置
    temp = points[0]
    points[0] = points[index]
    points[index] = temp




    # 前半部分是出发点 ；  后半部分是经过按斜率排序之后的n-1个坐标点      注意： “+”是拼接的含义，不是数值相加
    points = points[:1] + sorted(points[1:], key=compute)









    '''
    # 计算两个向量之间的叉积。返回三点之间的关系：

    <0，说明第三个点是向左转，则保留第二个点（栈顶元素）
    >0，说明第三个点是向右转，则删除第二个点（栈顶元素）
    =0，说明三点共线

    '''
    def ccw(a, b, c):
        return (b[2] - a[2]) * (c[1] - b[1])   -    (c[2] - b[2]) * (b[1] - a[1])




    # 用列表模拟一个栈。（最先加入的是前两个点，前两次while必定不成立，从而将点加进去）
    convex_hull = []




    for p in points:
        '''
        # 如果能顺时针方向（右转）连接第三个顶点，就删除栈顶元素再加入这个顶点 ； 否则（向左转才达到第三个顶点），直接加入这个顶点
        convex_hull[-2]：栈顶元素下面的元素
        convex_hull[-1]：栈顶元素
        p：要分析的第三个顶点

        '''
        while len(convex_hull) > 1 and ccw(convex_hull[-2], convex_hull[-1], p) >= 0:
            convex_hull.pop()
```

```
            convex_hull.append(p)

    return convex_hull




if __name__ == '__main__':
    points = read_file('input.txt')
    hull = Graham_Scan(points)
    for i in hull:
        print(i[0])
```

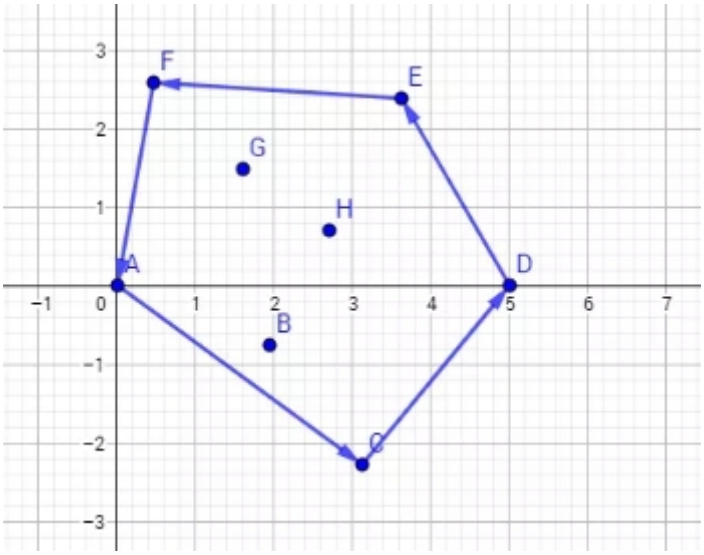
input.txt内容

```
1,3.6,2.4
2,5,0
3,3.1,-2.2
4,2.8,0.8
5,0,0
6,1.7,1.5
7,0.4,2.6
8,1.9,-0.8
```

输出：（代表连接顺序）

```
[5, 0.0, 0.0]
[3, 3.1, -2.2]
[2, 5.0, 0.0]
[1, 3.6, 2.4]
[7, 0.4, 2.6]
```

大致的图像为：



> https://ysw1912.github.io/post/algorithm/2d_convex_hull/
>
> https://en.wikipedia.org/wiki/Graham_scan