分治法求解思路

1.找出由横坐标最大、最小的两个点p1 p2所组成的直线。用该直线将点集分成上下两set1，set2部分。

上半区分治：

2.分别从set1、set2找出与线段p1p2构成的面积最大的三角形的点p3,p4。

3.从set1中找出在直线p1p3左侧的点集leftset1、在直线p3p2右侧的点集rightset1。

4将leftset1,leftset2重复2、3步骤，直至找不到在直线更外侧的点。

下半区分治：

5.从set2找出在直线p1p4左侧的点集leftset2、在直线p3p4右侧的点集rightset2。

6.将leftset1,leftset2重复2、3步骤，直至找不到在直线更外侧的点。

合并：

其实在合并之前，答案就已经生成了，只不过答案点集是无须的，现在让他按顺时针有序

点与直线的位置判断

可通过以下行列式的正负值判断直线与点之间的位置关系，同时数值为点与线段所围成的三角形的面积：

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3$$

下面的`绘制过程`动态图，可以加深理解。思想就是递归地把整体分为两半，递归地为每一个半区找一个点使之与分界线组成的三角形面积最大，直到在这个半区的分界线以外找不出点再组成三角形，则这个半区停止，回溯到上一个半区继续处理...

代码

```
import random
import matplotlib.pyplot as plt
```

```
import matplotlib.animation as animation

draw_line_lists = []

# 根据三角形的三个顶点坐标，计算三角形面积
def calc_area(a, b, c):
    x1, y1 = a
    x2, y2 = b
    x3, y3 = c
    return x1 * y2 + x3 * y1 + x2 * y3 - x3 * y2 - x2 * y1 - x1 * y3


# 在（start,end)区间内，随机生成具有 n 个点的点集（return: list [(x1,y1)...(xn,yn)]）
def sample(n, start=0, end=101):
    return list(zip([random.randint(start, end) for _ in range(n)],
[random.randint(start, end) for _ in range(n)]))




def Up(left, right, points, borders):
    """
    寻找上半部分边界点
    :param left: tuple, 最左边的点
    :param right: tuple, 最右边的点
    :param points: 所有的点集
    :param borders: 边界点集
    :return:
    """

    # 画图用，记录处理步骤
    draw_line_lists.append(left)
    draw_line_lists.append(right)


    # left 和 right_point两个点构成了一个直线，现在想在这条直线上面寻找一个点，要求构成的三角形面积最大
    area_max = 0   # 记录最大三角形的面积

    for item in points:
        if item == left or item == right:   # 构成直线的两个点，也在lists集合里，但他不在这条直线的上面，不对他计算
            continue
        else:
            temp = calc_area(left, right, item)  # 暂存该点与直线构成的面积，用于接下来的迭代比较
            if temp > area_max:
                max_point = item    # 记录当前构成最大三角形的那个点
                area_max = temp     # 记录当前最大三角形面积

    if area_max != 0:                # 这里其实就是递归边界。当一条线的上面不再有点可以试探，就停止，返回到上一层，处理他的兄弟节点的子树。
        borders.append(max_point)
        Up(left, max_point, points, borders)      # 原来的左边界点不变，将刚找到的构成最大三角形得到点作为右边界点，继续递归
        Up(max_point, right, points, borders)     # 原来的右边界点不变，将刚找到的构成最大三角形得到点作为左边界点，继续递归
```

```
def Down(left, right, points, borders):
    """
    同Up中的参数
    """
    # 画图用，记录处理步骤
    draw_line_lists.append(left)
    draw_line_lists.append(right)


    # left 和 right_point两个点构成了一个直线，现在想在这条直线下面寻找一个点，要求构成的三角形面积最大
    area_max = 0    # 记录最大三角形的面积

    for item in points:
        if item == left or item == right:    # 构成直线的两个点，也在lists集合里，但他不在这条直线的下面，不对他计算
            continue
        else:
            temp = calc_area(left, right, item)  # 暂存该点与直线构成的面积，用于接下来的迭代比较
            if temp < area_max:
                max_point = item     # 记录当前构成最大三角形的那个点
                area_max = temp      # 记录当前最大三角形面积

    if area_max != 0:                # 这里其实就是递归边界。当一条线的下面不再有点可以试探，就停止，返回到上一层，处理他的兄弟节点的子树。
        borders.append(max_point)
        Down(left, max_point, points, borders)      # 原来的左边界点不变，将刚找到的构成最大三角形得到点作为右边界点，继续递归
        Down(max_point, right, points, borders)     # 原来的右边界点不变，将刚找到的构成最大三角形得到点作为左边界点，继续递归




# 合并步骤。执行到这里时，分治已经结束，答案已经生成，这个函数的作用就是把无序的答案按照顺时针的顺序整理一下
def order_border(points):
    """
    :param points: 无序边界点集
    :return: list [( , )...( , )]
    """
    points.sort()    # 按x轴顺序先排一下，用来寻找最左边和最右边的点
    first_x, first_y = points[0]  # 最左边的点
    last_x, last_y = points[-1]   # 最右边的点
    up_borders = []      # 上半边界
    dowm_borders =[]     # 下半边界
    # 对每一个点进行分析
    for item in points:
        x, y = item
        if y > max(first_y, last_y):     # 如果比最左边和最右边点的y值都大，说明一定在上半区
            up_borders.append(item)
```

```
        elif min(first_y, last_y) < y < max(first_y, last_y):    # 如果比最左边和最右边点的y值中间，就要借助三角形的面积来做判断。如果面积为负，说明是一个倒置的三角形，即第三个点在直线的下方，即下半区；否则为上半区
            if calc_area(points[0], points[-1], item) > 0:
                up_borders.append(item)
            else:
                dowm_borders.append(item)
        else:                               # 如果比最左边和最右边点的y值都小，说明一定在下半区
            dowm_borders.append(item)

    list_end = up_borders + dowm_borders[::-1]  # 最终顺时针输出的边界点
    return list_end



def draws(points, list_frames, gif_name="save.gif"):
    """
    生成动态图并保存
    :param points: 所有点集
    :param list_frames: 帧 列表
    :param gif_name: 保存动图名称
    :return: .gif
    """
    min_value = 0
    max_value = 100

    all_x = []
    all_y = []
    for item in points:
        a, b = item
        all_x.append(a)
        all_y.append(b)

    fig, ax = plt.subplots()  # 生成轴和fig，  可迭代的对象
    x, y = [], []  # 用于接受后更新的数据
    line, = plt.plot([], [], color="red")  # 绘制线对象，plot返回值类型，要加逗号

    def init():
        # 初始化函数用于绘制一块干净的画布，为后续绘图做准备
        ax.set_xlim(min_value - abs(min_value * 0.1), max_value + abs(max_value * 0.1))  # 初始函数，设置绘图范围
        ax.set_ylim(min_value - abs(min_value * 0.1), max_value + abs(max_value * 0.1))
        return line

    def update(points):
        a, b = points
        x.append(a)
        y.append(b)
        line.set_data(x, y)
        return line

    plt.scatter(all_x, all_y)  # 绘制所有散点
    ani = animation.FuncAnimation(fig, update, frames=list_frames, init_func=init, interval=1500)   # interval代表绘制连线的速度，值越大速度越慢
    ani.save(gif_name, writer='pillow')
```

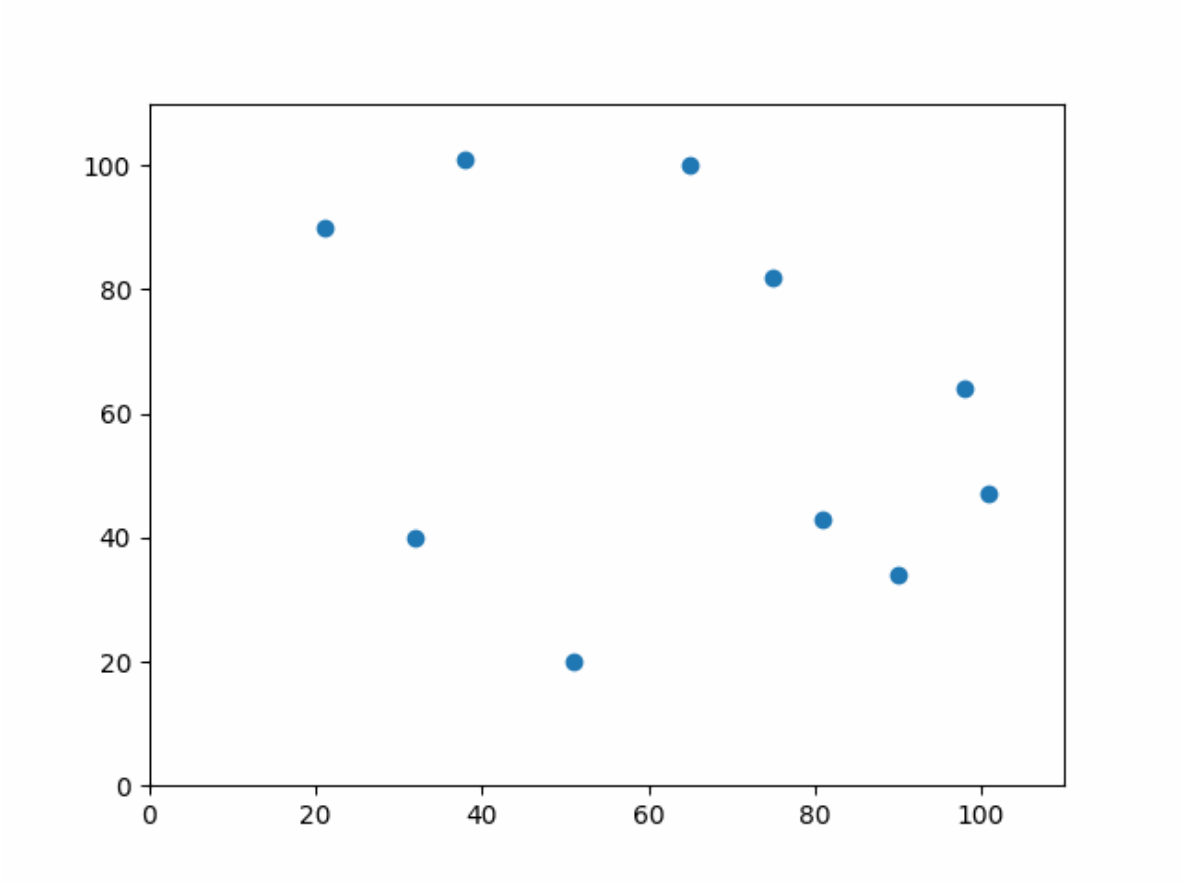
```python
def show_result(points, results):
    """
    画图
    :param points: 所有点集
    :param results: 所有边集
    :return: picture
    """
    all_x = []
    all_y = []
    for item in points:
        a, b = item
        all_x.append(a)
        all_y.append(b)

    for i in range(len(results)-1):
        item_1=results[i]
        item_2 = results[i+1]
        #  横坐标,纵坐标
        one_, oneI = item_1
        two_, twoI = item_2
        plt.plot([one_, two_], [oneI, twoI])
    plt.scatter(all_x, all_y)
    plt.show()



if __name__ == "__main__":

    points = [(101, 47), (32, 40), (21, 90), (65, 100), (98, 64), (81, 43), (51, 20), (75, 82), (90, 34), (38, 101)]
    # points = sample(100)   # 随机生成点

    points.sort()   # 先按x轴排序一下，用来寻找最左边和最右边的点
    borders = []  # 边界点集
    Up(points[0], points[-1], points, borders)    # 上边界点集
    Down(points[0], points[-1], points, borders)  # 下边界点集
    borders.append(points[0])
    borders.append(points[-1])  # 将首尾两个点添加到边界点集中
    results = order_border(borders)  # 顺时针边界点
    print(results)  # 顺时针输出答案

    results.append(results[0])  # 把最后一个点和源点连起来，绘制成闭合连线（仅在画图时这样处理）

    show_result(points,results)   # 显示静态结果
    draws(points, results, "result.gif")  # 绘制动态结果
    draws(points, draw_line_lists, "process.gif")  # 绘制动态过程
```

程序运行结果

输入坐标点

```
[(101, 47), (32, 40), (21, 90), (65, 100), (98, 64), (81, 43), (51, 20), (75, 82), (90, 34), (38, 101)]
```

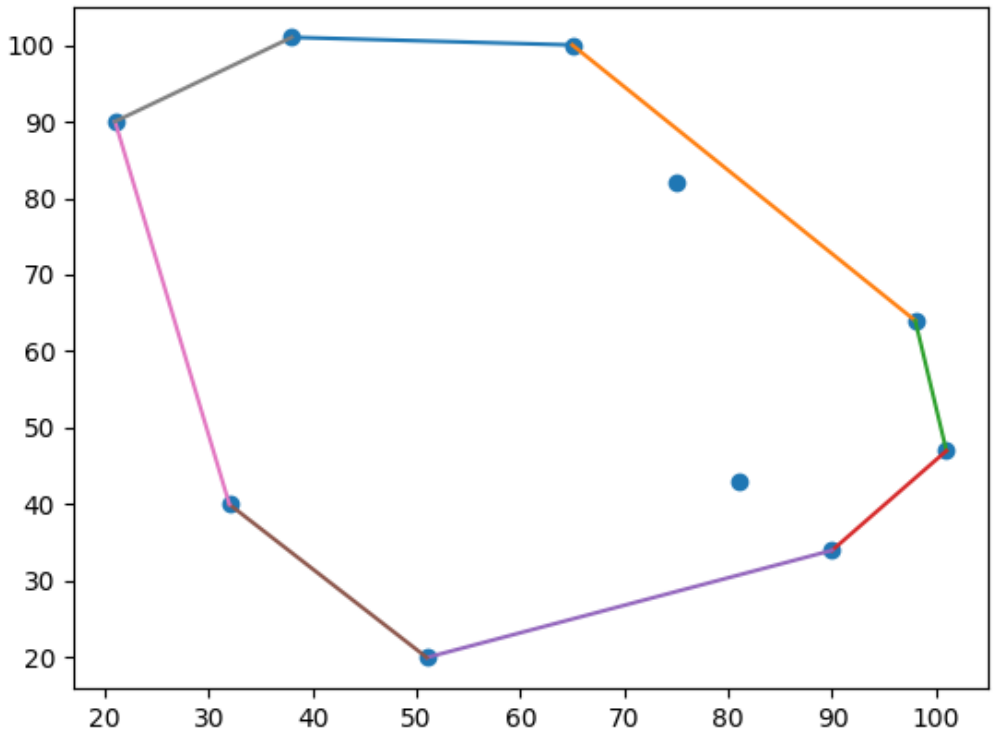
输出动态绘制过程



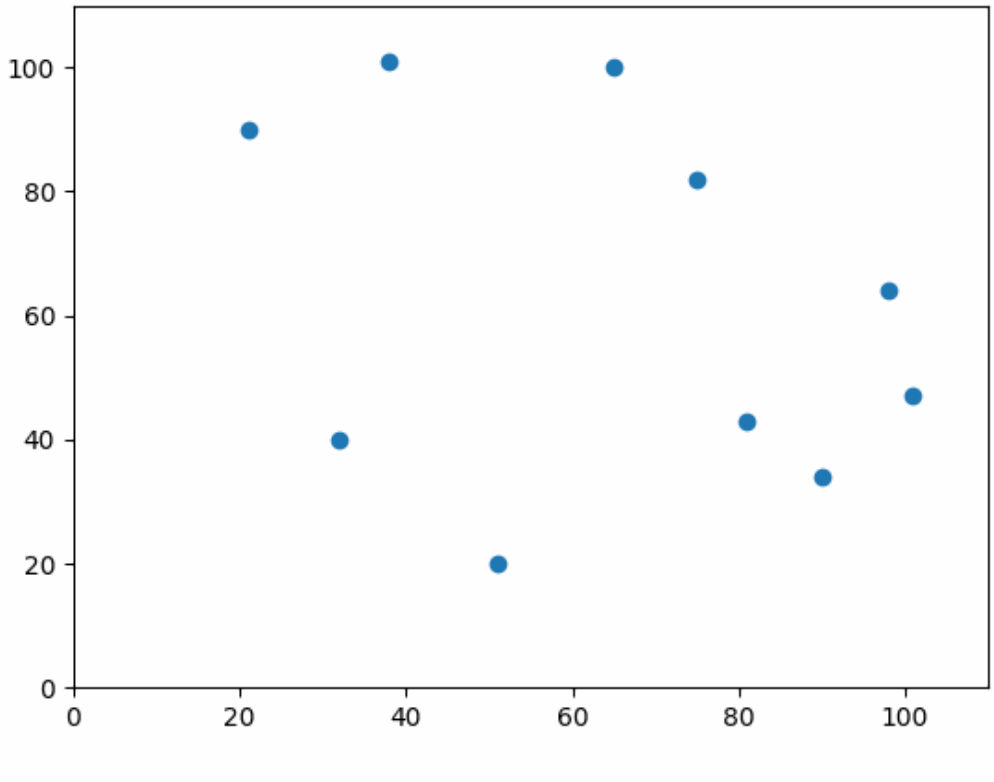
输出连接点顺序

```
[(38, 101), (65, 100), (98, 64), (101, 47), (90, 34), (51, 20), (32, 40), (21, 90)]
```

静态输出最大凸多边形


100
90
80
70
60
50
40
30
20
20
30
40
50
60
70
80
90
100

动态输出最大凸多边形


100
80
60
40
20
0
0
20
40
60
80
100