hw-5

0413

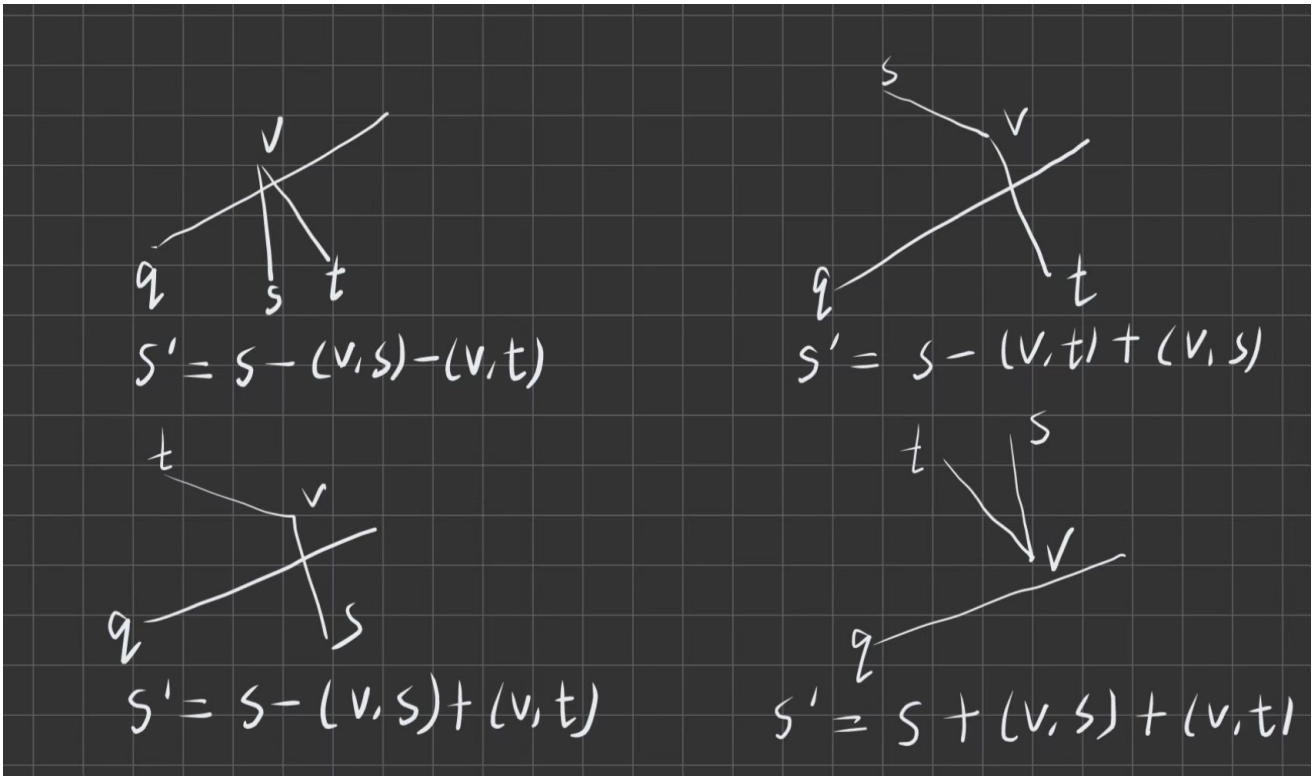
1. 设P是包围在给定矩形R中的一个简单多边形，q为R中任意一点，设计高效算法寻找连接q和R外部一点的线段，使得该线段与P相交的边的数量最少

Solution :

将该多边形表示为用邻接表存储的图 G = (V, E)，设有 n 个顶点 n 条边，图中每个顶点的度数均为 2。以 q 为坐标原点建立极坐标系，求出每个顶点的极坐标 (ρ,θ)，并按极角 θ 的大小排序。使用扫描线算法，记S为当前射线与P中相交的边的集合，注意到S的元素发生改变只可能出现在扫描的射线经过某个点时，具体而言还可细分为以下四种情况：

记v为扫描线即将经过的节点，s和t是v在多边形中相邻的两个节点

- $(v,s)\in S,(v,t)\in S$,旋转过v后$S=S-(v,s)-(v,t)$
- $(v,s)\notin S,(v,t)\in S$,旋转过v后$S=S-(v,t)+(v,s)$
- $(v,s)\in S,(v,t)\notin S$,旋转过v后$S=S-(v,s)+(v,t)$
- $(v,s)\notin S,(v,t)\notin S$,旋转过v后$S=S+(v,t)+(v,s)$



因此，解决思路为将所有顶点的极角值存储在一个小顶堆中，使用扫描线算法依次扫描，每次扫描到一个点时利用该点在图中相邻两条边的信息来更新集合S和当前射线与多边形相交边数**count**,(集合S和count的初始化是通过$O(N)$时间遍历整张图找到所有与极轴，即$\theta = 0$相交的边)，在遍历过程中记录**count**的最小值与对应的角度，最终射线即为最小**count**对应的角度，在该射线上任取R外一点即满足题设。总时间复杂度为$O(NlogN)$,主要来自于扫描线算法遍历小顶堆的时间消耗。

2.(1) 给定平面上一组点，已知每个点的坐标，求最远点对之间的距离，即点集的直径；(2) 如果已知任意两点之间的距离d_{ij}，存储在矩阵D中，求这组点的直径，此时一种直观的解法就是把D扫描一遍，选择其中最大的元素即可。由于距离d_{ij}满足非负性、对称性和三角不等式，我们可以给出一种时间亚线性的近似算法。算法很简单，由原来确定性算法的检查整个矩阵改为只随机检查D的某一行，这样时间复杂性就由原来的$O(n^2)$减少为$O(n)$，相对于输入规模n^2而言，这是一个时间亚线性的算法。证明时间代价减小的同时，解不会小于最优值的一半

Solution :

(1)

引理：最远点对一定在这组点的凸包上

证明：反证法

假设最远点对为 (p,q)，其中 q 不在这组点的凸包上，则根据凸包的定义 q 一 定在凸包的内部。对凸包多边形进行三角形分划，一定能找到

- 凸包上一点x,使得q在线段px上，则pq不是最远点对
- 或一个三角形$\triangle pxy$,使得q在$\triangle pxy$内部

  因为三角形内角和为 180°，在 $\triangle qxy$ 中，$\angle xqy < 180^\circ$，所以 $\angle pqx + \angle pqy > 180^\circ$。不妨设 $\angle pqx \le \angle pqy$，则有 $2\angle pqy > 180^\circ$，即 $\angle pqy > 90^\circ$

  则$\triangle pqy$为钝角三角形且py为钝角所对边，从而$py > pq$,即pq不是最远点对

于是最远点对一定在凸包上

应用Graham扫描算法可以找到这组点的凸包，时间复杂度为$O(nlogn)$

作凸多边形两条平行的支撑线，并沿逆时针方向同时旋转两条平行支撑线。则若凸包上两点是最远点对，一定存在某一时刻，使两点均在平行线上。因此在旋转的过程中求出能同时出现在两平行线上的点对之间的距离，并找到最大值即可。 因为两条支撑线将共同遍历全部的点一次，所以算法的时间复杂度为$O(n)$。

具体而言：

1. 计算多边形 *y* 方向上的端点。我们称之为 *ymin* 和 *ymax*
2. 通过 *ymin* 和 *ymax* 构造两条水平切线。由于他们已经是一对对踵点，计算他们之间的距离并维护为一个当前最大值
3. 同时旋转两条线直到其中一条与多边形的一条边重合
4. 一个新的对踵点对此时产生。计算新的距离，并和当前最大值比较， 大于当前最大值则更新
5. 重复步骤3和步骤4的过程直到再次产生对踵点对 *(ymin,ymax)*
6. 输出确定最大直径的对踵点对

总的时间复杂度为$O(nlogn)$

(2)

证明：在矩阵D中任取一行，即在图中任取一点x，考察x到其余n-1个点的距离

记(p,q)为距离最大点对，即d_{pq}是直径

- 若x=p或x=q，则解为最优值
- 否则，考察边px与边qx，由三角不等式，一定有

$$d_{px}+d_{qx}\geq d_{pq}$$

等号成立当且仅当p,q,x共线且x位于p,q之间

因此$max\{d_{px},d_{qx}\}\geq\frac{1}{2}d_{pq}$,从而解不会小于最优解的一半

3. 有n种液体$S_1,S_2,\ldots,S_n$，都含有A,B两种成分，含量分别为{a1,b1},{a2,b2},⋯,{an,bn}，ai+bi<100%。现欲利用这n种液体配制目标液体T，使之A和B的含量分别为x和y。设计算法判别能否成功配制，并给出算法时间复杂性。

Solution :

- 思路1：若(x,y)可成功由(ai,bi)配置，则其一定能表示为若干个(ai,bi)的加权平均，从而(x,y)一定落在由所有点(ai,bi)构成的凸包内，因此可使用Graham算法找到这组点的凸包，并判断(x,y)是否在凸包内即可，总的时间复杂度为$O(nlogn)$
- 思路2：若(x,y)在(ai,bi)构成的凸包内，任取凸包上一个点v对凸包做三角形剖分，(x,y)一定落在某个三角形内或者某条从v出发的边上。若在三角形内，则(x,y)可由这个三角形三个顶点的加权平均表示；若在边上，则(x,y)可由边的两个端点的加权平均表示。因此，可以T=(x,y)为坐标原点，到任意一点S=(ai,bi)为x轴建立直角坐标系。之后遍历所有的点，找出x轴上方一点P，使得$\angle PTS$最大，找出x轴下方一点Q，使得$\angle QTS$最大，之后判断$\angle PTQ$是否大于180度即可。(大于180度则可构成三角形，从而知T在凸包内可以表示。若存在负x轴上的点，则直接输出可以)

0420

1. 在平面上给定一个有n个点的集合S，求S所有的极大点。极大点的定义：设p1=(x1,y1)和p2=(x2,y2)是平面上的两个点，如果x1≤x2并且y1≤y2，则称p2支配p1，记为$p_1 \prec p_2$。点集S中的点p为极大点，意味着在S中找不到一个点q，q≠p并且$p \prec q$，即p不被S中其它点支配。

Solution：

注意到横坐标最大的点一定是极大点，为了找出所有的极大点，可以先对横坐标从大到小降序排序($O(nlogn)$)，之后按排好序的横坐标遍历所有点，并不断更新遍历过程中y坐标的最大值max_y(初始化为$-\infty$)，每次遇到一个新的点，将其纵坐标y与max_y相比较

- 若$y > max_y$，则横坐标比该点大的点中没有纵坐标比该点大的，从而该点为极大点，并更新max_y
- 若$max_y > y$，则该点一定被之前的某个点所支配，继续遍历即可

排序时间复杂度为$O(nlogn)$，遍历时间复杂度为$O(n)$，总的时间复杂度为$O(nlogn)$

2. 对凸多边形，1）有多少种三角划分的方法？2）如何使对角线长度之和最小？

Solution：

三角划分个数

假设对凸n边形($n \geq 3$)，三角划分的方法为g_n，对于边V_1V_n,任选一顶点V_k，向V_1和V_n连边。将三角形$V_1V_nV_k$分割出去，剩下两个多边形，一个多边形有顶点{1，2，3，…，k}，所以是k边形；另一个多边形有顶点{k，k+1,…,n}，所以是（n-k+1）边形。然后继续分割多边形直到都变成了三角形，这个过程可以用递归或递推实现。

首先选边V_1V_n，由于旋转对称，我们不管选哪条边，都可以看作V_1V_n，因此只有一种选法，则递推关系如下：

$$g_n = g_2 \cdot g_{n-1} + g_3 \cdot g_{n-2} + \cdots + g_{n-1} \cdot g_2 \quad (n \geq 3)$$

其中g_2定义为1

直接求解这个递推公式我们需要通过使用生成函数将其转化为两个函数的卷积，但相比之下有更简单的思路。注意到卡特兰数C_n的递推公式为

$$C_n = C_0 \cdot C_{n-1} + C_1 \cdot C_{n-2} + \cdots + C_{n-1} \cdot C_2 \quad (n \geq 1)$$

其中$C_0 = 1$

比较两式，形式基本相同，仅仅是下标的区别，因此考虑下标转换，注意到若令$g_n = C_{n-2}$，则初值$g_2 = C_0 = 1$，且恰好满足递推关系，由递推序列可唯一确定表达式，从而

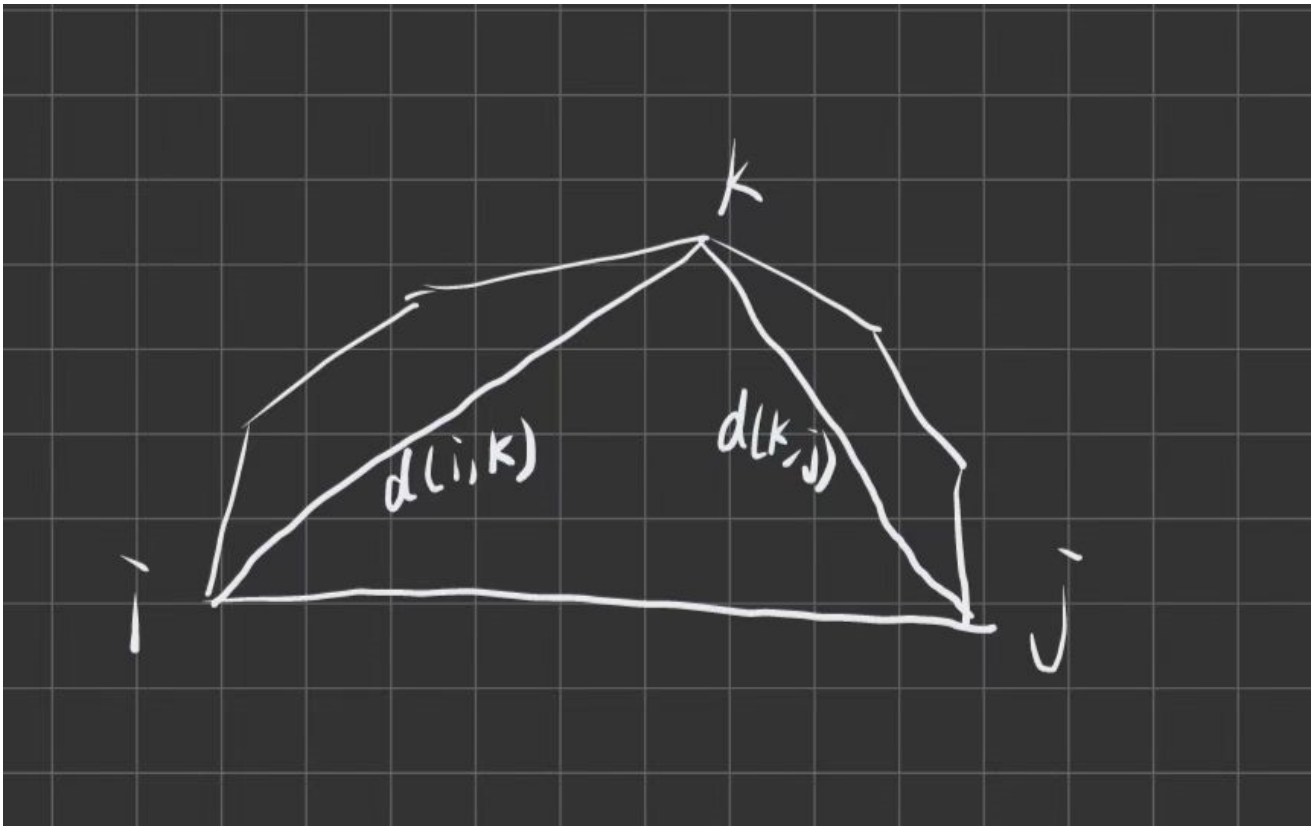
$$g_n = C_{n-2} = \frac{C_{2n-4}^{n-2}}{n-1}, \quad (n \geq 3)$$

使对角线长度之和最短

使用动态规划求解即可，仍然使用上题的递归方法，对于某次选择三角形$V_1V_nV_k$，剩下两个多边形(一个k边形，一个n-k+1边形)一定已经达到了对角线长度之和最小的状态，因此可递归求解子问题，令$dp[i][j]$表示连接第i个节点和第j个节点构成的子多边形的最短对角线长度之和，则状态转移式为

$$dp[i][j] = \min_k \{dp[i][k] + dp[k][j] + d(i,k) + d(k,j)\}$$

k是i和j之间的分点



通过动态规划即可递归求解问题，总的时间复杂度为$O(n^3)$(枚举起点，枚举终点，枚举分点)

3. 给定平面上n条线段，设计算法用O(nlogn)时间确定其中是否有两条线段相交

Solution：

使用扫描线算法，事件列表为线段的所有端点按横坐标升序，当遇到第一个交点时算法结束。交点出现时，相交的两线段一定是相邻的。时间复杂度为$O((2n)log(2n)) = O(nlogn)$

伪代码如下：

```
输入：L={l_i}，线段集合
输出：是否存在交点
对所有线段的两个端点以横坐标为键，(x,y,l)为值建立最小化堆H
S={}    #按纵坐标升序的最小化堆
for (x,y,l) in H:            #按横坐标升序遍历
    if l not in S:
        index = S.push(l)
        return True if intersect(l, s[index-1]) or intersect(l,
s[index+1])
        #插入新线段时检查相邻线段是否相交
    else:
        index = S.find(l)
        return True if intersect(s[index-1], s[index+1])
        #移除线段时检查相邻线段是否相交
        S.remove(l)
return False
```

4. 用扫描线算法求解最近邻点对问题

Solution：

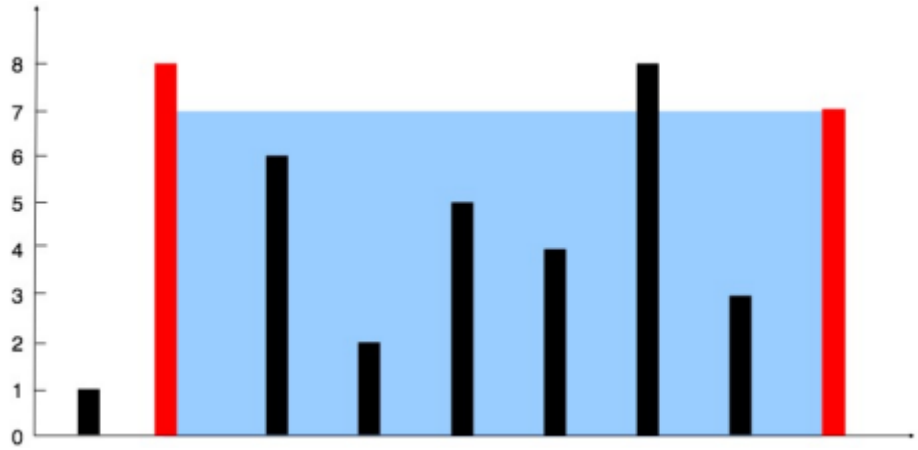
事件为所有的点，按横坐标升序排序，扫描线状态为已经扫描过且到扫描线的距离小于某个值的所有点

```
输入：{(xi,yi)}
输出：p,q(距离最近的两个点)
将所有点以横坐标为键，建立最小堆Hx
Q=queue(), Hy=Redblacktree(key=y)
```

```
#初始化扫描线状态，使用队列存储
#同时使用红黑树存储扫描线状态中的点，便于查找
d=INF
#距离初始化为无穷
for (x,y) in Hx:
    while (x_,y_) <- Q.top() and |x-x_|>d:  #最左侧的点到扫描线的距离超过了d，直接移除
        Q.pop((x_,y_)), Hy.remove((x_,y_))
    #对于新的事件，先更新扫描线状态
    for (x_,y_) in Hy and |y-y_|<d:   #使用红黑树查找满足条件的点，这样的点最多不超过6个
        if ||(x,y)-(x_,y_)|| < d:
            d = ||(x,y)-(x_,y_)||
            q=(x,y), q=(x_,y_)
    Q.push((x,y)), Hy.push((x,y))
```

思路就是在扫描线扫描的过程中不断更新扫描线状态中的点(移除不可能满足条件的点)，同时在给定y范围内查找有可能提供更小距离的点，这样的点一定不会超过6个，用这种方法扫描按x坐标排序之后的数组即可

5. 给定正整数a1，a2，…，an，代表n条线段（由点(i,ai)和(i,0)构成，i=1,2,⋯,n），从中找出两条线段，使之与x轴构成的容器能够包含尽可能多的水



Solution：

所求水量为

$$\max_{i<j}\{\min\{ai, aj\} \cdot (j-i)\}$$

直接遍历时间复杂度为$O(n^2)$，可使用更简单的算法：

使用双指针，分别指向最左端和最右端，从两端开始向中间靠拢，如果左端线段短于右端，那么左端右移，反之右端左移，直到左右两端移到中间重合，记录这个过程中每一次组成木桶的容积，返回其中最大的。时间复杂度为$O(n)$

合理性解释：当左端线段L小于右端线段R时，选择将L右移，此时舍弃的是L与右端(R-1,R-2,...)这些线段构成的木桶。这些线段是没有必要判断的，因为这些木桶的容积一定没有L和R组成的木桶大

```
int maxArea(int[] height)
{
    if (height.length < 2) return 0;
    int ans = 0;
    int l = 0;
    int r = height.length - 1;
    while (l < r) {
        int v = (r - l) * Math.min(height[l], height[r]);
        if (v > ans) ans = v;
        if (height[l] < height[r]) l++;
        else r--;
    }
    return ans;
}
```