2020 年 春 季学期研究生课程考核

实验报告（一）

考　核　科　目：高级算法设计与分析

学生所在院（系）：计算机科学与技术学院

学 生 所 在 学 科：计算机科学与技术

学　生　姓　名：于晟健

学　　　　　号：19S003037

学　生　类　别：全日制学术型硕士研究生

考　核　结　果　　　　　　　　　　阅　卷　人

实验 1　分治算法

1.1 实验目的

（1）掌握分治算法的设计思想与方法；

（2）熟练使用高级编程语言实现分治算法；

（3）通过对比简单算法以及不同的分治求解思想，理解算法复杂度；

1.2 实验学时

4 学时。

1.3 实验问题

求解凸包问题：输入是平面上 n 个点的集合 Q，凸包问题是要输出一个 Q 的凸包。其中，Q 的凸包是一个凸多边形 P，Q 中的点或者在 P 上或者在 P 中。

1.4 实验步骤

1.4.1 实现基于枚举方法的凸包求解算法

考虑 Q 中的任意四个点 A、B、C、D，如果 A 处于 BCD 构成的三角形内部，那么 A 一定不属于凸包 P 的顶点集合。由凸包性质可知，具有最小 x 值的点一定在凸包顶点中，故算法代码如下：

```python
def enumeration(Q: set):
    if len(Q) <= 3: return Q
    Qlist = list(Q)
    m = Qlist[0]
    # the leftmost point must be in the convex hull
    for n in Qlist:
        if n.x < m.x: m = n
    # P includes the points which are not in the convex hull
    P, N = set(), len(Q)
    P.add(m)
    for i in range(N - 2):
        a = Qlist[i]
        if a in P: continue
        for j in range(i + 1, N - 1):
            b = Qlist[j]
            if b in P: continue
            for k in range(j + 1, N):
                c = Qlist[k]
                if c in P: continue
                # if a in the triangle(b,c,m) then add a in P
                if isInTriangle(a, b, c, m):
                    P.add(a)
                    break
                # if b in the triangle(a,c,m) then add b in P
                if isInTriangle(b, a, c, m):
                    P.add(b)
                    break
                # if c in the triangle(b,a,m) then add c in P
                if isInTriangle(c, b, a, m):
                    P.add(c)
            # if a in P then select the next possible point
            if a in P: break
    P.remove(m)
    return Q - P
```

其中，函数 $isInTriangle(p, a, b, c)$判断点 p 是否在点 a、b 和 c 组成的三角形中（若四点共线，则在三角形中），算法代码如下：

```python
def isInTriangle(p: point, a: point, b: point, c: point):
    """ ... """
    # count the sign of Triangle
    signOfTrig = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x)
    if not signOfTrig: return False
    # count the sign of p with a,b a,c and b,c
    signOfAB = (b.x - a.x) * (p.y - a.y) - (b.y - a.y) * (p.x - a.x)
    if signOfAB * signOfTrig < 0: return False
    signOfCA = (a.x - c.x) * (p.y - c.y) - (a.y - c.y) * (p.x - c.x)
    if signOfCA * signOfTrig < 0: return False
    signOfBC = (c.x - b.x) * (p.y - c.y) - (c.y - b.y) * (p.x - c.x)
    if signOfBC * signOfTrig < 0: return False

    return True
```

算法的时间复杂度为 $O(n^3)$。

1.4.2 实现基于 *Graham-Scan* 的凸包求解算法

当沿着凸包逆时针漫游时，总是向左转；在极坐标系下按照极角大小排列，然后逆时针方向漫游点集，去除非凸包顶点（非左转点）。因此，我们选择具有最小 y 值的最右点 p_0 作为出发点，按照与 p_0 极角的大小逆时针方向排序其余点（若两点与初始点具有相同的极角，则由远及近进行排列）；排序完成后，我们利用栈来进行非左移动的判断，算法如下：

```python
def grahamScan(Q: set):
    P = set()
    Qlist, N = list(Q), len(Q)
    if N < 3: return P
    # choose the point which have the min of y as the start point
    for i in range(N):
        if Qlist[i].y < Qlist[0].y:
            Qlist[i], Qlist[0] = Qlist[0], Qlist[i]
            continue
        # if points have the same y then choose the point which have the max x
        if Qlist[i].y == Qlist[0].y and Qlist[i].x > Qlist[0].x:
            Qlist[i], Qlist[0] = Qlist[0], Qlist[i]
    quickSort(Qlist, 1, N - 1)

    stack = []
    stack.append(Qlist[0])
    stack.append(Qlist[1])
    stack.append(Qlist[2])
    for i in range(3, N):
        stack_len = len(stack)
        top, next_top = stack[stack_len - 1], stack[stack_len - 2]
        a = vector(Qlist[i].x - next_top.x, Qlist[i].y - next_top.y)
        b = vector(top.x - next_top.x, top.y - next_top.y)
        # delete the top(stack) on the left of the line which from next_top(stack) to Qlist[i]
        while stack_len > 2 and cross(a, b) >= 0:
            stack.pop()
            stack_len = len(stack)
            top, next_top = stack[stack_len - 1], stack[stack_len - 2]
            a = vector(Qlist[i].x - next_top.x, Qlist[i].y - next_top.y)
            b = vector(top.x - next_top.x, top.y - next_top.y)
        stack.append(Qlist[i])

    P = set(stack)
    return P
```

其中，*quickSort*(*Qlist*, 1, *N*-1)表示对 *Qlist* 中第 2 个元素到第 *N* 个元素的快速排序算法：

```python
def quickSort(Qlist, low, high):
    if low < high:
        i = low - 1
        pivot = cos(vector(Qlist[high].x - Qlist[0].x, Qlist[high].y - Qlist[0].y))
        dis = distance(Qlist[high], Qlist[0])

        for j in range(low, high):
            if cos(vector(Qlist[j].x - Qlist[0].x, Qlist[j].y - Qlist[0].y)) > pivot:
                i = i + 1
                Qlist[i], Qlist[j] = Qlist[j], Qlist[i]
                continue
            # because we choose the min y with the possible max x as the start point
            # so we need sort the points which have the same cos value from far to near
            if cos(vector(Qlist[j].x - Qlist[0].x, Qlist[j].y - Qlist[0].y)) == pivot:
                if distance(Qlist[j], Qlist[0]) > dis:
                    i = i + 1
                    Qlist[i], Qlist[j] = Qlist[j], Qlist[i]
        Qlist[i + 1], Qlist[high] = Qlist[high], Qlist[i + 1]

        quickSort(Qlist, low, i + 1 - 1)
        quickSort(Qlist, i + 1 + 1, high)
```

s

算法时间复杂度为 $O(nlogn)$。

1.4.3 实现基于分治思想的凸包求解算法

（1）边界条件：若 $|Q|<3$，算法停止；若 $|Q|=3$，按照逆时针方向输出 $CH(Q)$ 的顶点；

（2）*Divide*：选择一个垂直于 x-轴的直线把 Q 划分为基本相等的两个集合 Q_L 和 Q_R，Q_L 在 Q_R 的左边；

（3）*Conquer*：递归地为 Q_L 和 Q_R 构造 $CH(Q_L)$ 和 $CH(Q_R)$；

（4）*Merge*：*a.* 找一个 QL 的内点 p；*b.* 在 $CH(Q_R)$ 中找与 p 的极角最大和最小顶点 u 和 v；*c.* 构造如下三个点序列：按逆时针方向排列的 $CH(Q_L)$ 的所有顶点；按逆时针方向排列的 $CH(Q_R)$ 从 u 到 v 的顶点；按顺时针方向排列的 $CH(Q_R)$ 从 u 到 v 的顶点；*d.* 合并上述三个序列；*e.* 在合并的序列上应用 *Graham-Scan*；故算法代码如下：

```python
def divideAndConquer(Qlist: list):
    N = len(Qlist)
    if N < 3: return Qlist
    flag = 1  # whether there is a line
    for i in range(N):
        if Qlist[i].x != Qlist[0].x:
            flag = 0
            break
    if N == 3 or flag:
        # choose the point which have the min of y as the first point
        for i in range(N):
            if Qlist[i].y < Qlist[0].y: Qlist[i], Qlist[0] = Qlist[0], Qlist[i]
        quickSort(Qlist, 1, N - 1)
        return Qlist
    # Divide
    x_median = median(Qlist)
    # Conquer
    QL = divideAndConquer([q for q in Qlist if q.x <= x_median])
    QR = divideAndConquer([q for q in Qlist if q.x > x_median])
    # Merge
    if len(QR) > 0:
        QR_ymax, tip = QR[0], 0
        for q in range(0, len(QR)):
            if QR_ymax.y < QR[q].y:
                QR_ymax, tip = QR[q], q
            else:
                break
        # QR[0] denotes u and QR[tip] denotes v
        new_Q = QL + QR[:tip + 1] + QR[len(QR) - 1:tip:-1]
    else:
        new_Q = QL

    return list(grahamScan(set(new_Q)))


def divideConquer(Q: set):
    Qlist = list(Q)
    P = divideAndConquer(Qlist)

    return set(P)
```

算法时间复杂度为 $O(nlogn)$。

1.4.4 实现随机生成点集合 Q 的算法

实现随机生成正方形(0,0)-(0,100)-(100,100)-(100,0)内的点集合 Q 的算法；

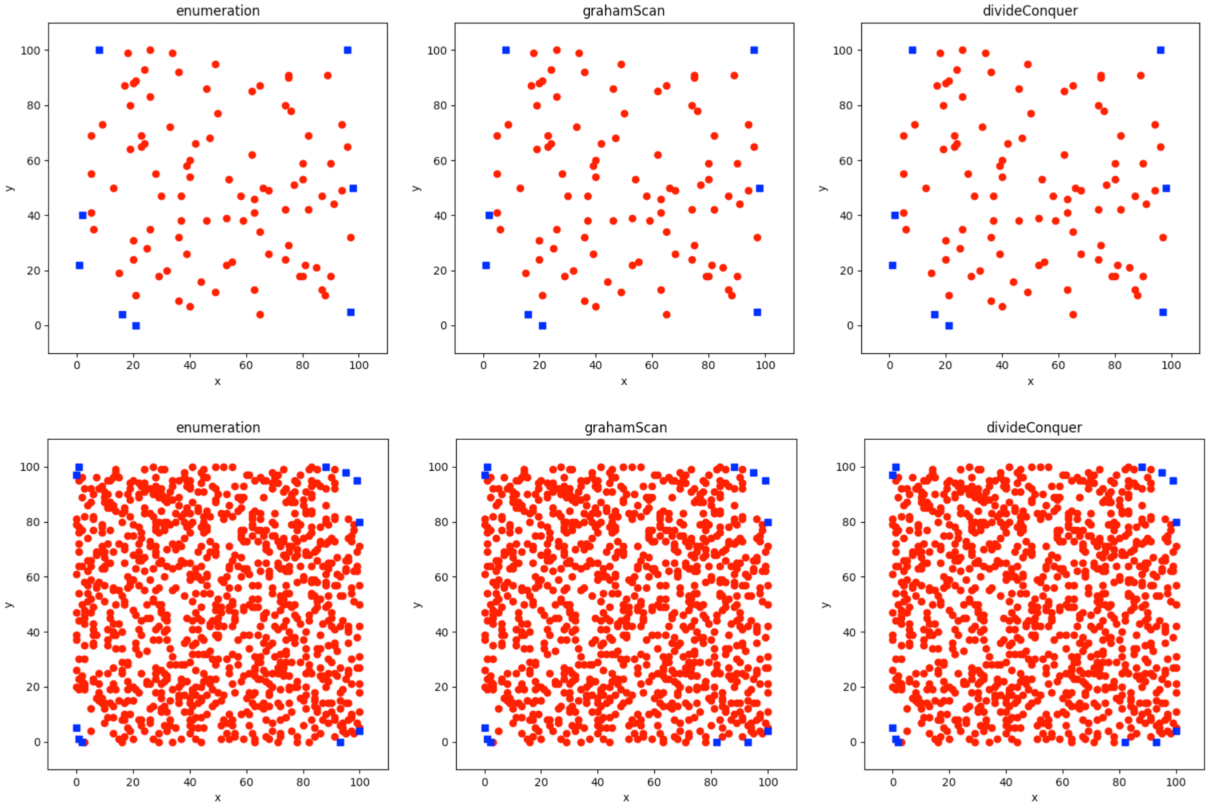
```python
def pointSet(N: int, limit=100):
    """

    :param N: the number of points
    :param limit: the restrict of points' area
    :return: set Q which includes N points
    """
    Q = set()
    while len(Q) < N:
        a = point(random.randint(0, limit), random.randint(0, limit))
        Q.add(a)  # if point a is in Q before added, Q will no change
    return Q
```

1.5 实验结果和分析

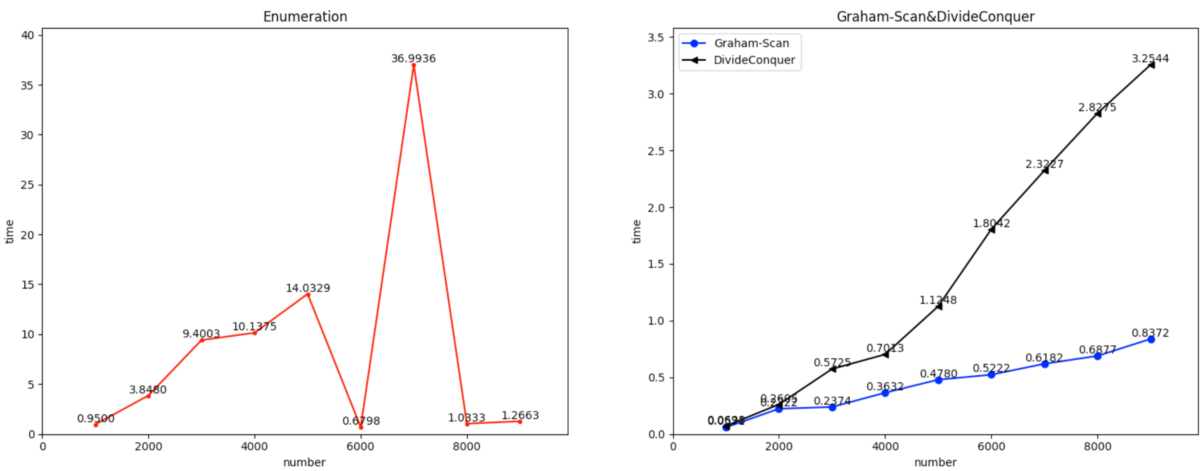
1.5.1 实验结果

利用点集合生成算法自动生成大小不同数据集合，如点数大小分别为(1000, 2000, 3000, ...)的数据集合；对每个算法，针对不同大小的数据集合，运行算法并记录算法运行时间；对每个算法，绘制算法性能曲线，对比算法；

首先，以点数 100 和 1000 为例，我们分别给出每个算法的运行结果，如下图所示：



其次，我们给出每个算法运行的性能曲线（横轴表示点数，纵轴表示运行时间），如下图所示：



其中，枚举算法求解凸包问题会出现震荡现象。这是因为若在枚举过程，当循环删

去足够多的点时会较为快速的完成；若循环遍历的点是根据坐标顺序排列的，则每次循环会删去较少的点，即循环较多。而在我们的算法中数据为随机排列，故会出现震荡的现象。

1.5.2 对比三种凸包求解算法

基于枚举的凸包求解算法每次选择三个点遍历，故算法的时间复杂度为 $O(n^3)$；基于 Graham-Scan 的凸包求解算法需要对所有点根据极角进行排序，故算法的时间复杂度为 O(nlogn)；基于分治算法的凸包求解算法在 Divide 阶段使用 O(n)算法求中值，满足递归式 T(n)=2T(n/2)+O(n)，故算法的时间复杂度为 O(nlogn)。

1.6 实验心得

通过本次实验更加深刻地理解了分治算法的设计思想与方法，同时也加深了对特殊情况的考虑（如极角相同，点共线等情况）。