哈尔滨工业大学

<<算法设计与分析>>

实验报告之一

(2014 年度秋季学期)

姓名：	王纪轩
学号：	**14S137050**
学院：	软件学院
教师：	骆吉洲 副教授
实验时间：	**2014.12.2**

实验一 分治算法

一、实验目的

- 掌握分治算法的设计思想与方法
- 熟练使用高级编程语言实现分治算法
- 通过对比简单算法以及不同的分治求解思想，体验分治算法

二、实验内容

- 实现基于枚举方法的凸包求解算法
- 实现基于Graham-Scan的凸包求解算法
- 实现基于分治思想的凸包求解算法
- 对比三种凸包求解算法

三、实验过程及结果

（一）枚举法（暴力法）：

暴力法比较简单，基本思想是对点集合里的任意四个点，如果其中一个点在另外三个点组成的三角形内部，则把这一个点删除，待将所有满足上述条件的点删除后，剩余的点即为凸包上的点。

在遍历的过程中，首先每次遍历的四个点之间不能重复，其次，如果某一遍历删除了一些点，那么在接下来的遍历中就不在考虑被删除的点。因此，遍历的过程中需要加一些判断。如果遍历任意四个点，那么需要四重循环，时间复杂度$T(n)=\Theta(n^4)$，但如果固定某一个点，遍历其余三个点，时间复杂度降为n^3。纵坐标最小的点显然肯定在凸包内，那么就固定这个点，遍历其余三个点。

在判断一个点是否在另外三个点组成的三角形内部的思路是：首先，如图 1-1 所示，两个点 A，B 将平面划分为两个部分，另外一点 P 带入直线方程得到 g(A,B,P)，P 在直线 AB 上则 g(A,B,P)=0，若 P 在 AB 一侧，则 g(A,B,P)大于零或小于零；那么，如果 P 在三角形 ABC 内部，只要满足图 1-2 所示条件即可。

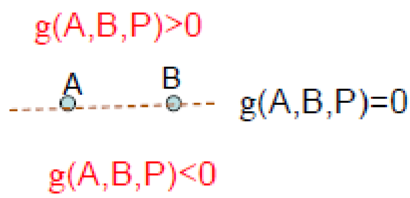


图 1-1 点与直线的位置关系

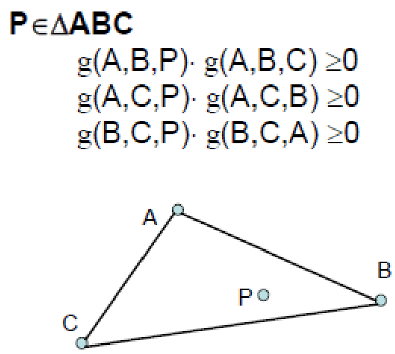


图 1-2 判断点在三角形内部

但这里需要注意的是 ABC 共线的问题，如果 ABC 共线，那么 P 点只要不在 ABC 两两组成的线段上即可，如图 1-3 所示。如果 ABC 共线且不这条线不垂直，那么 P 的横坐标在 ABC 横坐标的最大值和最小值之间，就可以删除 P。判断点是否在三角形内部的流程图如图 1-4 所示。

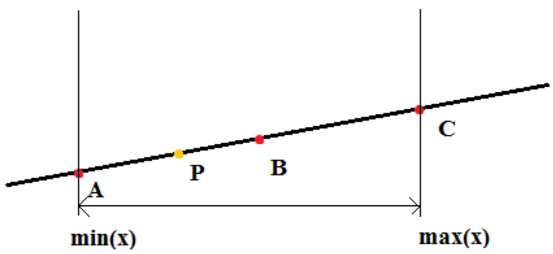


图 1-3 共线示意图

遍历结束后剩余点即为所有凸包上的点，但此时得到凸包上的点并非有序，还需要进一步排序处理：首先对所有点按照横坐标排序，同时可以得到横坐标最小和最大的点，记为 a 和 b，这两个点将所有点分成上下两个部分 A 和 B，则按照 a、顺序输出 A、b、逆序输出 B 即可得到凸包序列。注意在将点分为 A、B 两个部分时保证原顺序不变，这样只需要上述一次排序即可。如图 1-5 所示。

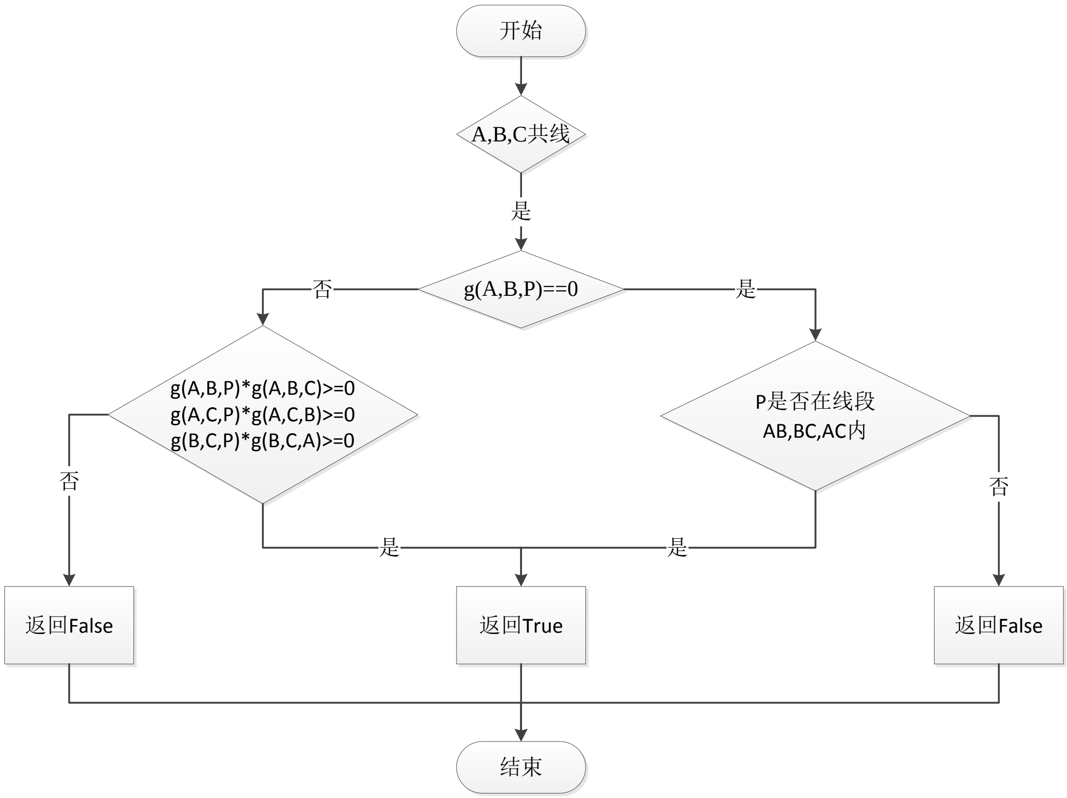


图 1-4 isInTriangle 流程图

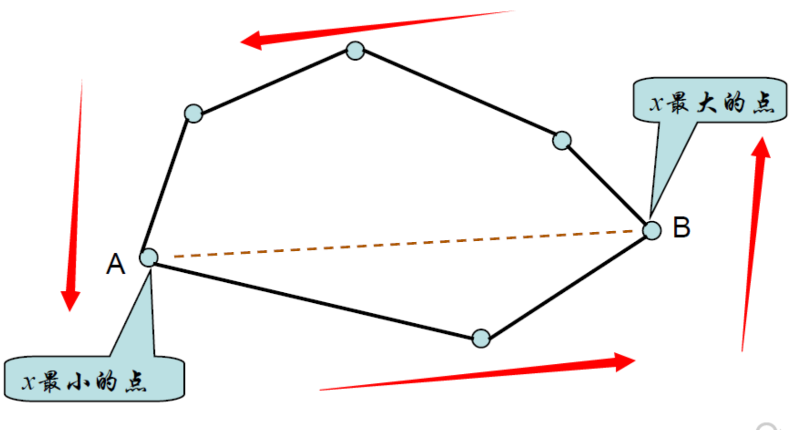


图 1-5 合并

综上所示，实现了凸包计算的暴力法，时间复杂度为：

$$T(n) = \Theta(n^3)$$

（二）Graham-Scan：

Graham-Scan 的基本思想是，当沿着凸包上的点逆时针漫游时，总是左转弯。如图 2-1 所示。将所有点按照极坐标排好序，逆时针遍历，除去非左转点，剩余即为凸包点。

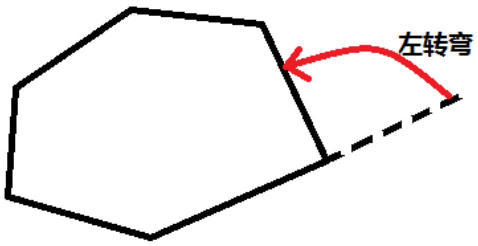


图 2-1 左转弯示意图

伪代码如下：

```
/*  栈S从底到顶存储按逆时针方向排列的CH(Q)顶点  */
1.  求Q中y-坐标值最小的点p0;
2.  按照与 p0 极角(逆时针方向)大小排序 Q 中其余点，结果为<p1, p2, …,pm>;
3. Push(p0, S);      Push(p1, S);      Push(p2, S);
4. FOR i=3 TO m DO
5.          While Next-to-top(S)、Top(S)和pi形成非左移动  Do
6.                     Pop(S);
7.          Push(pi, S);
8. Rerurn S.
```

说明如下：

1、Q 中 y-坐标值最小的点 p_0 肯定包含在凸包集合中，所以按照它将其他点的极角进行排序。以水平向右为 0 极角，这样保证了其余所有点的极角大小在 $(0,\pi)$ 范围内。

2、根据叉积来比较极角大小和判断转弯。叉积定义为：

$$\begin{aligned} p_1 \times p_2 &= \det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 \,. \end{aligned}$$

如果叉积大于零，则 p1 相对于 O 点位于 p2 的顺时针方向，如图 2-2 所示。它的几何含义就是 p1 在顺时针方向还是逆时针方向上更接近于 p2。

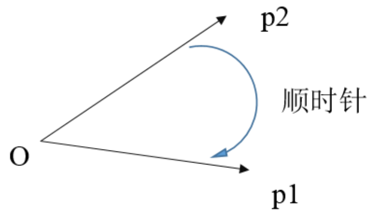


图 2-2 叉积示意图

可以使用叉积来对点进行极角排序，因为所有点都在 p_0 上方，故极角范围

为(0,π)，在这个范围中，如果某一点和 p_0 组成的向量 p_1 在另外一个点与 p_0 组成组成的向量 p_2 的顺时针方向，那么这个点的极角一定相对较小。如图 2-3 所示。同样，判断 B 点是否为非左转弯时，则判断 B 与 B 之前一点 A 组成的向量 *AB*，和 B 与之后的一点 C 做成的向量 *BC*，如果 *BC* 与 *AB* 的叉积大于等于零，则 C 点相对于 AB 为非左转弯（共线或者右转弯），这时 B 点肯定不为凸包内的点。如图 2-4 所示。

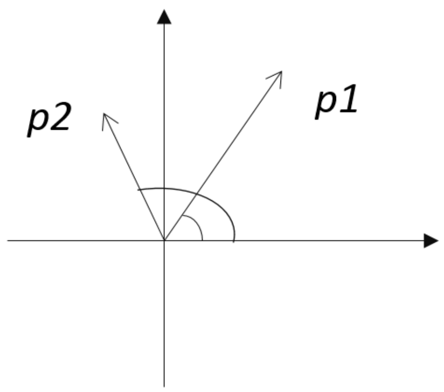


图 2-3 使用叉积比较极角

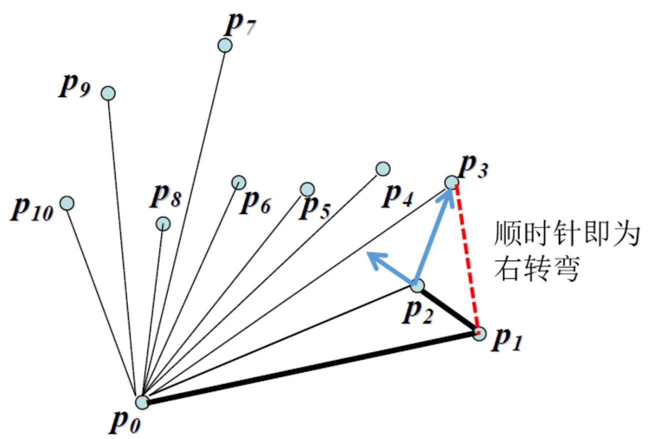


图 2-4 使用叉积判断转弯

3、每次遇到左转弯点，就压入栈中，如果遇到非左转点，则弹出栈顶点直到栈顶点为左转弯点。当将所有点扫描完毕时，栈中元素即为凸包点，按照从栈底到栈顶的顺序输出元素即可得到凸包序列。

综上所述，Graham-Scan 的主要运行时间在于极角的排序运算，我使用了快速排序的算法，故最终该算法的时间复杂度为：

$$T(n) = n\log n$$

（三）分治法：

凸包计算的分治法相对于上面两种算法来说复杂很多，有许多需要注意的细节。我基本上是按照课件上提供的分治思想来进行设计的，但在某些问题上采取了不同的策略。我的思路是：

1、Devide：对于一个点集，找出横坐标最大值和最小值，取平均值，横坐标小于等于这个平均值的为一组，大于这个平均值得为另一组。按照此规则不断进行划分。

2、Devide 结束：当划分至某组为两个点或者一个点时不再划分，并直接返回。如果某一组中点为多个横坐标相同的点，如果继续划分，下次划分这几个点仍然会被划分至一组，这样会进入无限递归。那么只返回**纵坐标最大和最小**的两个点即可，因为显然其他点不在凸包上。

3、Merge：合并的过程中，两边点集分别记为 Q_L，Q_R，左边部分已经返回凸包点集合 Q_L，并且 Q_L 中所有点按照逆时针排序，Q_R 同样如此，如图 3-1 所示。注意，只要满足点是**逆时针**顺序即可，**不需要以最低点为起始点**。另外，对于两个不同点来说，它们的位置关系**肯定是逆时针**的（或者肯定是顺时针的）并且我们认为它们是凸包集合的点，所以划分刚结束时直接返回的一个或两个点同样满足上述条件（逆时针，并且是凸包点）。

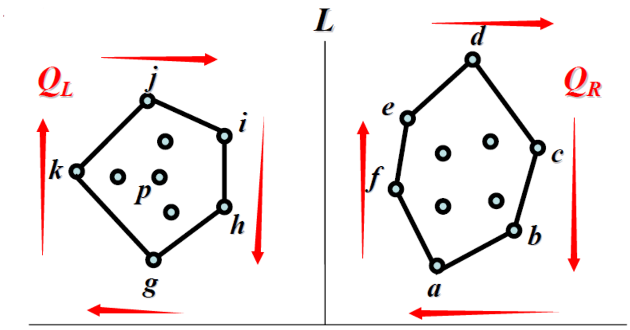


图 3-1 合并开始

合并的过程中，首先需要 Q_L 中心一点 p，该点并不一定需要实际存在，它只要在左边凸包内部即可，因此我取 p 的横坐标为 Q_L 中所有点横坐标的平均值，p 的纵坐标也为 Q_L 中所有点的纵坐标平均值。这样 p 点肯定在 Q_L 内部。下面将对 Q_L，Q_R 中的点相对于 p 点进行 Graham-Scan 算法，以找到 Q_L，Q_R 的凸包。

分治算法的关键在于不需要排序，如果现在对 QL，QR 中的点对 p 点进行极角排序，那么每一次合并都需要 nlogn 的时间，这样时间复杂度递推公式为：

$$T(n)=2T(n/2)+O(n\log n)$$

那么它的时间复杂度肯定会超过 Graham-Scan 的 *nlogn*，而我们使用分治算法的目的就是找到一个和 Graham-Scan 时间复杂度相同的算法，因此我们要避免排序，就像归并排序一样，每次合并的过程只需要线性时间，如果每次合并都重新排序那就得不偿失了。

上面说到，Q_L，Q_R 中的点都是**逆时针**顺序的，所以我们的目标就是，把两个逆时针顺序的点集合在**线性时间**内合并成一个逆时针顺序的点集合，然后就可以使用 Graham-Scan 了。如下图所示，取 p 点垂直方向为极角 0 轴，那么<h,i,j,k,g>为极角递增顺序；对于 Q_R，a，d 分别为**极角最小**和**极角最大**的两个点，那么<a,b,c,d>和<f,e>均为极角增大顺序。如图 3-2 所示。

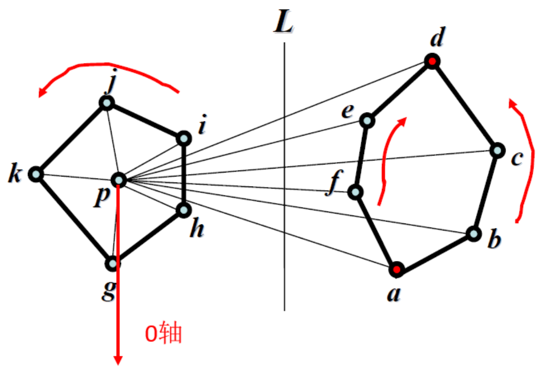


图 3-2 极角排序示意

由于每次递归完成后得到的序列不一定相对于 p 点是按照垂直方向为 0 轴极角递增的，比如我得到的序列 Q_L 是<k,g,h,i,j>，那我怎么才能把他们的顺序调整好，莫非又要排序？前面提到过，每次递归完，Q_L 和 Q_R 都是**逆时针**有序的，因此 Q_L 中的点之间的相对顺序是正确的，只不过极角最小点需要按照垂直方向为 0 度的标准进行调整。从 k 开始扫描，当遇到下一点比当前点极角小的时候，那下一个点就是极角最小点，如图 3-3 所示，k 点极角比 g 小，就把 k 移至序列末尾；g 点极角比 h 大，那将 g 点移至末尾，这样极角最小点 h 来到了队列首位，整个序列就按照递增顺序**移动**好了。这样，最坏的 情况是扫描了所有的点，所以时间复杂度为线性时间，达到了我们的目的。

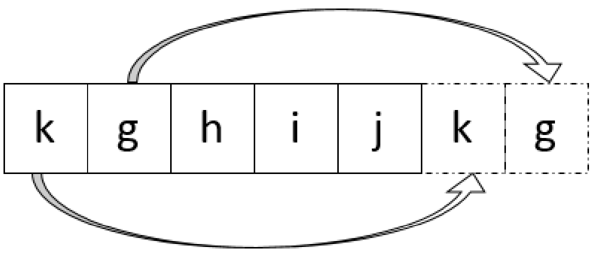


图 3-3 顺序调整示意

对于 Q_R 中的点也类似，首先找到极角最小和极角最大的点 a 和 d，然后顺序循环（到达末尾后从头开始）输出 a-d，逆序循环（到达开头后从末尾开始）输出 f-e，这样<a,b,c,d>和<f,e>成为了两个极角递增的序列。

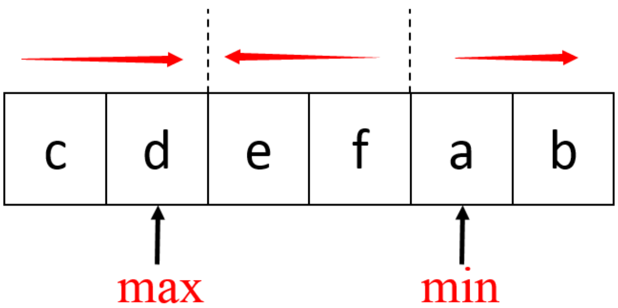


图 3-4 Q_R 调整示意

现在我们有了三条有序序列，参照归并排序中合并的思想，取三个指针分别指向三条序列<h,i,j,k,g>，<a,b,c,d>和<f,e>，找出最小的一个加入合并序列中，并将最小元素对应指针向后移，重复上述过程，直至合并完成。最终就得到了一条有序序列：<h,a,b,f,c,e,d,i,j,k,g>，“排序”工作至此完成（**但实际我们并没使用排序算法，仅仅进行了移位操作**），时间复杂度为线性复杂。

在进行下一步之前，我想说明一下我是怎样比较两点极角大小的，为了解释方便，我们规定 x 正方向为极角为 0 的方向， 如图 3-5 所示，显然图中三个点的极角顺序为 A>B>C。首先，比较三个点的象限，象限大的极角大，A 和 B 为第一象限，C 为第四象限，那么 C 比 A、B 大；相同象限比较三角函数值，比如 sin，因为 sin 在每个象限都是单调的，比如在第一象限，极角越大，sin 值越大。直接进行三角函数值得求解设计到开方和浮点运算，影响计算效率。如果 sin$b$>sin$a$，则有：

$$\frac{y_b}{\sqrt{{x_b}^2+{y_b}^2}}>\frac{y_a}{\sqrt{{x_a}^2+{y_a}^2}}$$

化简为：

$$(y_b x_a - x_b y_a)(y_b x_a + x_b y_a) > 0$$

这样只要满足上面两个括号内的值同号即可得到 B>A，这样就避免了浮点运算和整数溢出的麻烦。

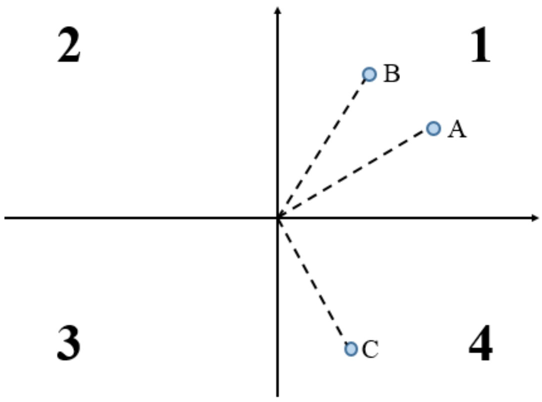


图 3-5 极角比较示意

到目前为止，我们找到了一个中心点 p，又得到了一个有序序列<h,a,b,f,c,e,d,i,j,k,g>，下面就可以进行 Graham-Scan 了！基本的步骤已经在上文中提到了，但是，注意我们的初始点是**虚构**的点 p，所以最终 p 点应被舍弃；另外，不同于上文所述，我们现在要进行(0,2π)上的扫描，结束的时候有一点需要注意。按照正常的步骤，去掉非左转点后得到一个序列，如图 3-6 所示，从 A 开始逆时针扫描，经过 B 及之后的一系列点，最终到达 C，按照上文的思路 Graham-Scan 到此为止了，但是由下图我们看出，对于 C-A-B 来说，A 是个右转弯点，A 不属于凸包上的点。所以，扫描到最后一个点 C 后，算法不能就此停止，应该继续循环执行，遇到像 A 这样的点就去除，直到遇到左转弯点 B，这时候扫描才真正结束。所以栈底若干元素可能会被删除，我们只要记录删到了第几个点（对于这图删除了第 1 个点，即为 A），然后输出从这个点往上，直到栈顶的点，即为当前点集的凸包（并且是逆时针顺序）。

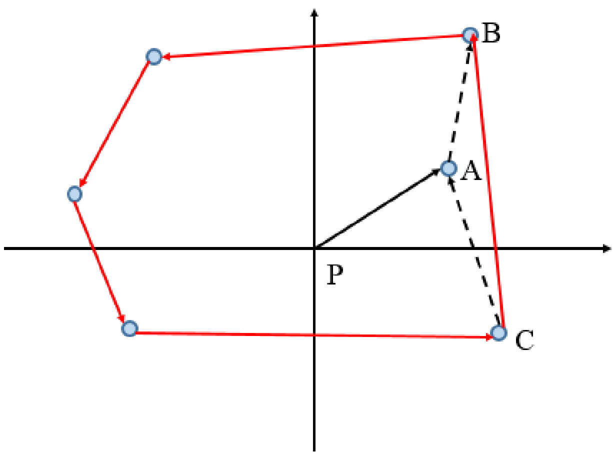


图 3-6 扫描结束示意

好了，现在合并过程终于完成了。我们得到的结果是，对于 Q_L 和 Q_R 上的点逆时针顺序的凸包，返回这个集合，就可以进行上一层次的合并了。重复这个过程，直至合并完所有的划分，最终整个点集的凸包就得到了！时间复杂度递推公式为：T(n)=2T(n/2)+O(n)，时间复杂度为 T(n)= O(nlogn)。

图 3-7 所示的流程图对上述过程进行了总结。

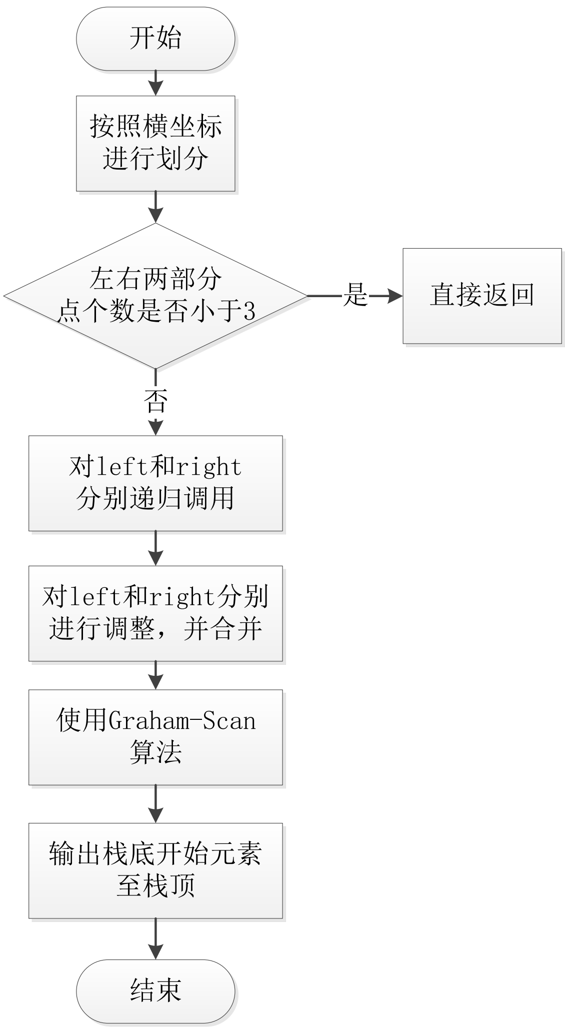


图 3-7 分治法流程图

（四）实现结果

由于本次实验重在实现算法，故在写程序的时候界面设计得很简单，程序运行如图 4-1 所示：

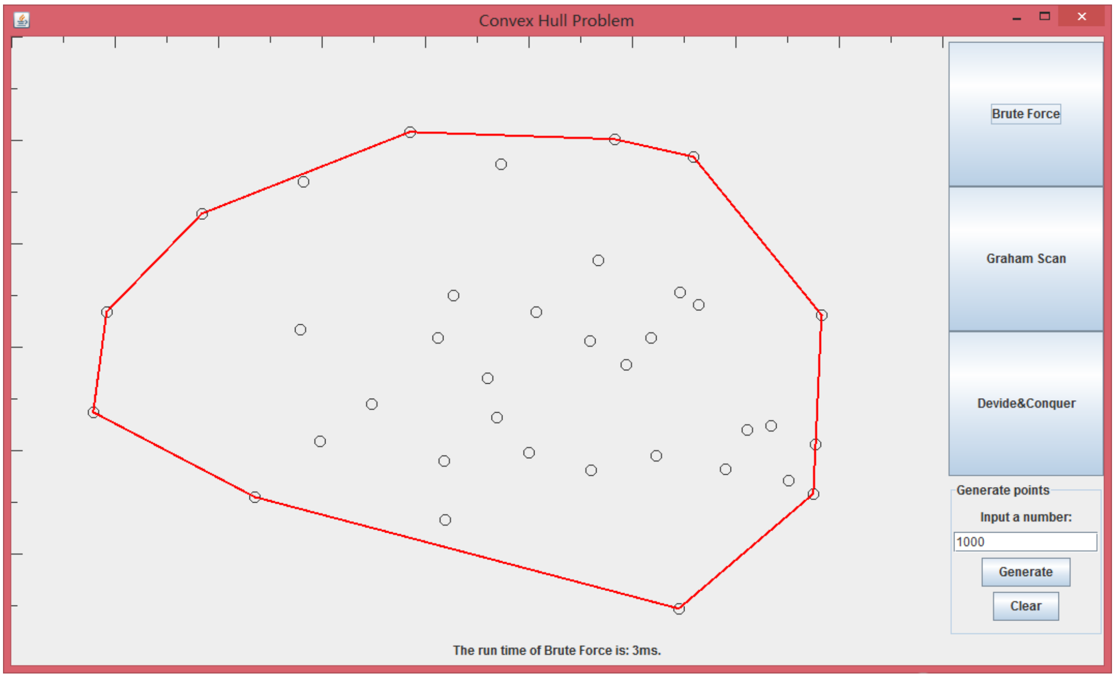


图 4-1 程序运行界面

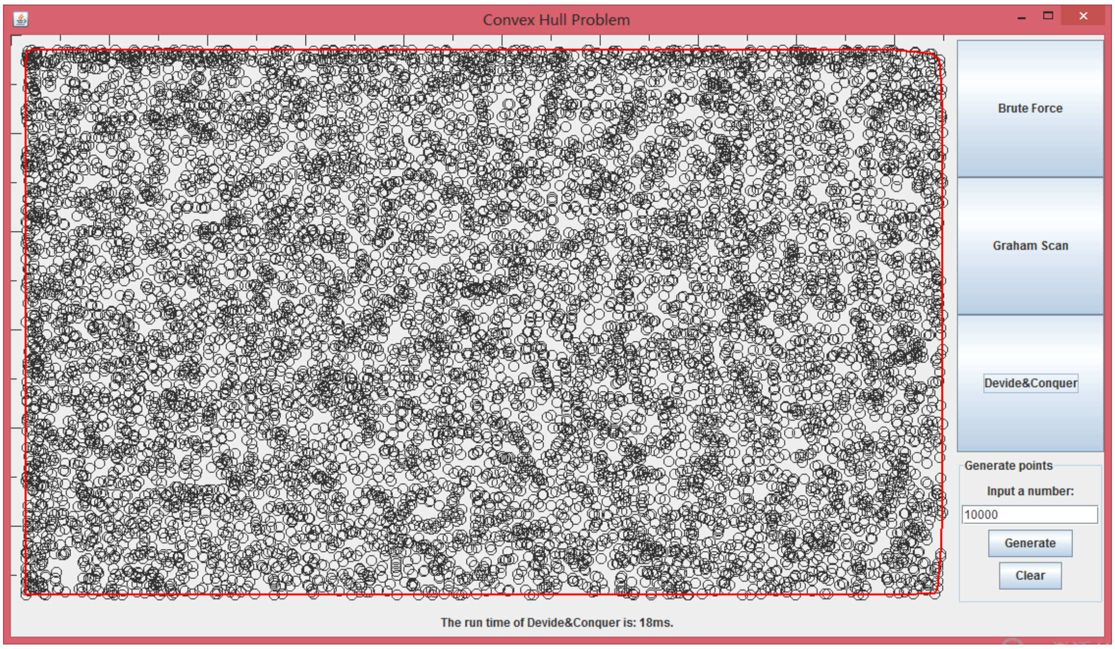


图 4-2 10000 个点运行界面

实现的主要功能有：

1、鼠标单击画点，双击已有点可以删除该点；

2、右侧前三个按钮，点击后分别用暴力法、Graham-Scan 法和分治法对面板里的点进行凸包计算；

3、在右侧下方输入框内输入数字，点击 Generate 按钮，可在面板上随机生成对应个数的点，点击 Clear 可清除所有点；

4、下方文字显示了算法的运行时间，以毫秒为单位。

本次实验使用 MyEclipse 开发，工程截图如图 4-3 所示。其中类 ConvexHull 包含主函数入口。DrawFram 为最外层组件，MyPanel 为主面板，DrawCompoment 实现了点和凸包的绘制。BruteForce、GrahamScan 和 DevideConquer 分别实现了暴力法、Graham-Scan 法和分治方法。部分代码截图如图 4-4 所示。

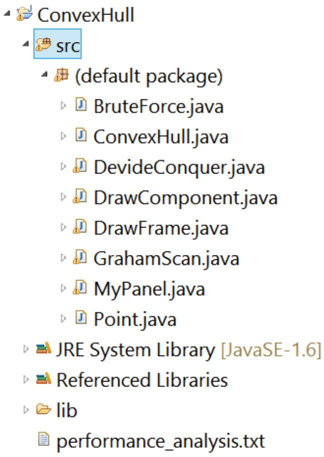
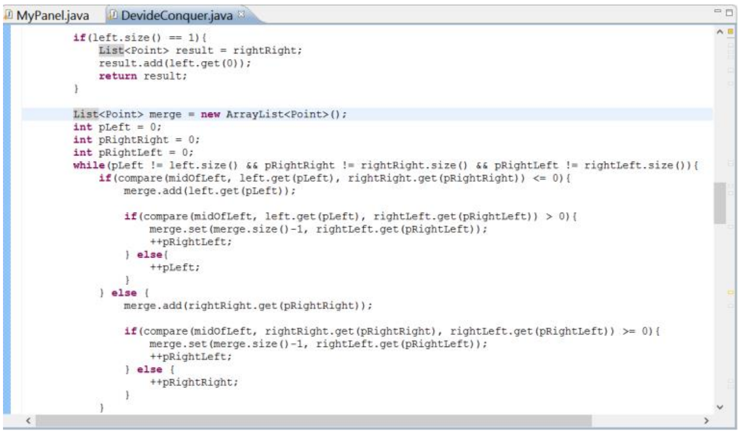


图 4-3 工程截图

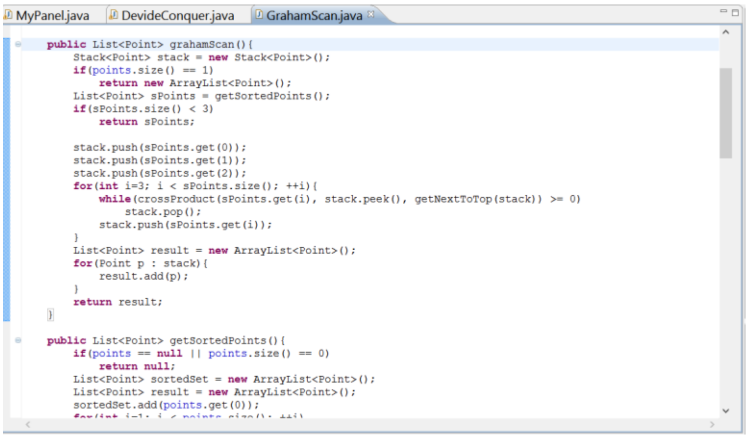



```java
        if(left.size() == 1){
            List<Point> result = rightRight;
            result.add(left.get(0));
            return result;
        }

        List<Point> merge = new ArrayList<Point>();
        int pLeft = 0;
        int pRightRight = 0;
        int pRightLeft = 0;
        while(pLeft != left.size() && pRightRight != rightRight.size() && pRightLeft != rightLeft.size()){
            if(compare(midOfLeft, left.get(pLeft), rightRight.get(pRightRight)) <= 0){
                merge.add(left.get(pLeft));

                if(compare(midOfLeft, left.get(pLeft), rightLeft.get(pRightLeft)) > 0){
                    merge.set(merge.size()-1, rightLeft.get(pRightLeft));
                    ++pRightLeft;
                } else{
                    ++pLeft;
                }
            } else {
                merge.add(rightRight.get(pRightRight));

                if(compare(midOfLeft, rightRight.get(pRightRight), rightLeft.get(pRightLeft)) >= 0){
                    merge.set(merge.size()-1, rightLeft.get(pRightLeft));
                    ++pRightLeft;
                } else {
                    ++pRightRight;
                }
            }
```




```java
    public List<Point> grahamScan(){
        Stack<Point> stack = new Stack<Point>();
        if(points.size() == 1)
            return new ArrayList<Point>();
        List<Point> sPoints = getSortedPoints();
        if(sPoints.size() < 3)
            return sPoints;

        stack.push(sPoints.get(0));
        stack.push(sPoints.get(1));
        stack.push(sPoints.get(2));
        for(int i=3; i < sPoints.size(); ++i){
            while(crossProduct(sPoints.get(i), stack.peek(), getNextToTop(stack)) >= 0)
                stack.pop();
            stack.push(sPoints.get(i));
        }
        List<Point> result = new ArrayList<Point>();
        for(Point p : stack){
            result.add(p);
        }
        return result;
    }

    public List<Point> getSortedPoints(){
        if(points == null || points.size() == 0)
            return null;
        List<Point> sortedSet = new ArrayList<Point>();
        List<Point> result = new ArrayList<Point>();
        sortedSet.add(points.get(0));
```

图 4-4 部分代码截图

（五）性能分析

在性能分析的时候，在(0,0)-(0,1000)-(1000,1000)-(1000,0)生成若干点，然后分别使用三种方法进行计算；对于每种算法重复生成点 30 次并计算每次的运行时间，取所得到的 30 个运行时间的平均值作为最终运行时间。其中前 20000 个点以前以 1000 递增，20000 后以 10000 递增。整体运行时间统计图如图 5-1 所示。可以看出，暴力法的运行效率远远低于另外两种的运行效率，而 Graham-Scan 和分治法相差不多。并且暴力法的增长的速度快很多。

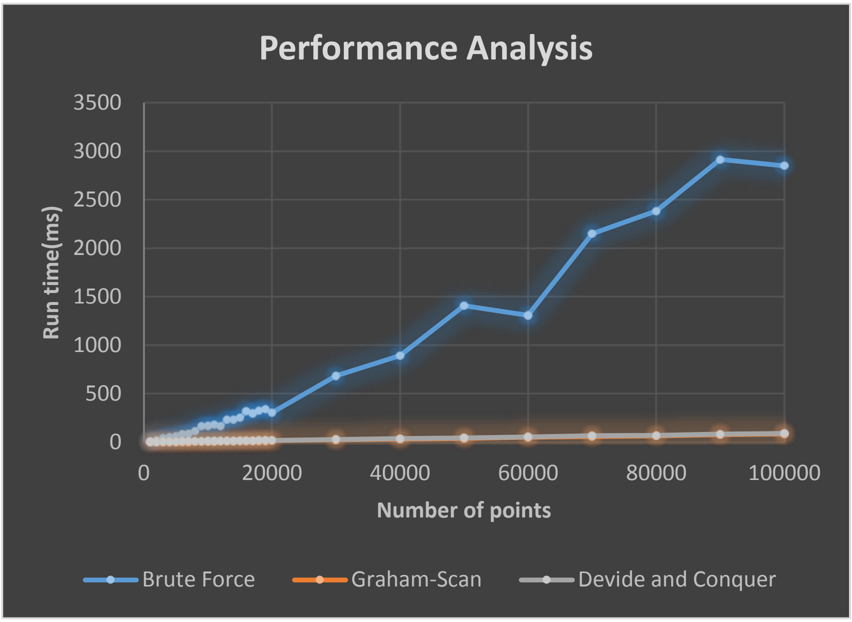


图 5-1 三种方法运行效率对比

1000-20000 并以 1000 递增的运行时间统计图如图 5-2 所示。

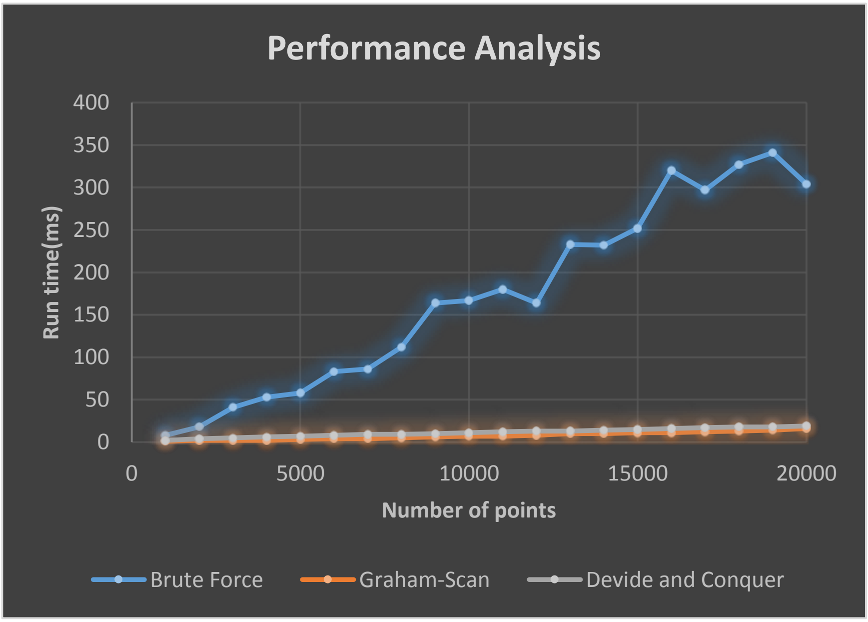


图 5-2 20000 个以内的点时间统计

Graham-Scan 和分治算法的对比如图 5-3 和图 5-4 所示，其中 5-4 为 1000-20000 按照 1000 递增的顺序画出的。可以看出两者的时间复杂度接近并且增长趋势相同。

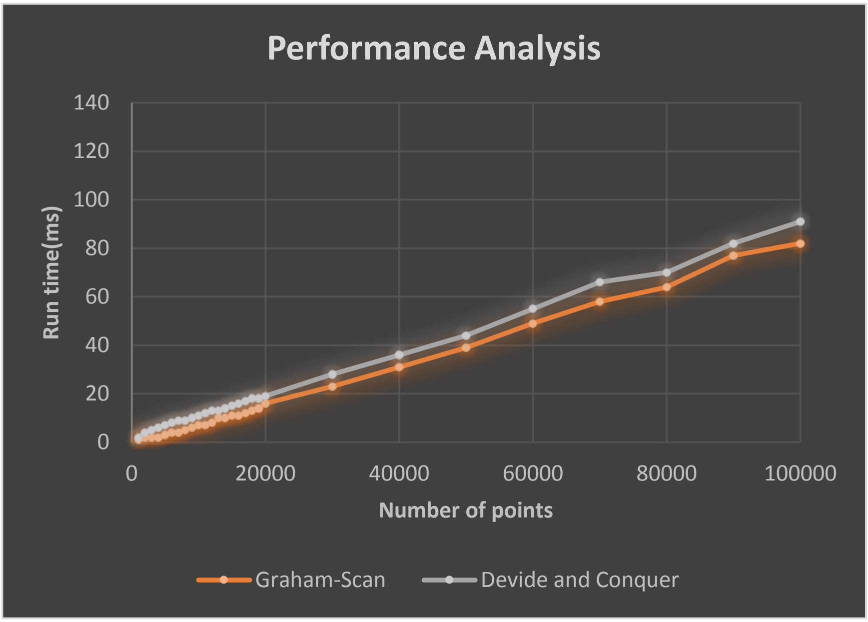


图 5-3 Graham-Scan 和分治算法对比

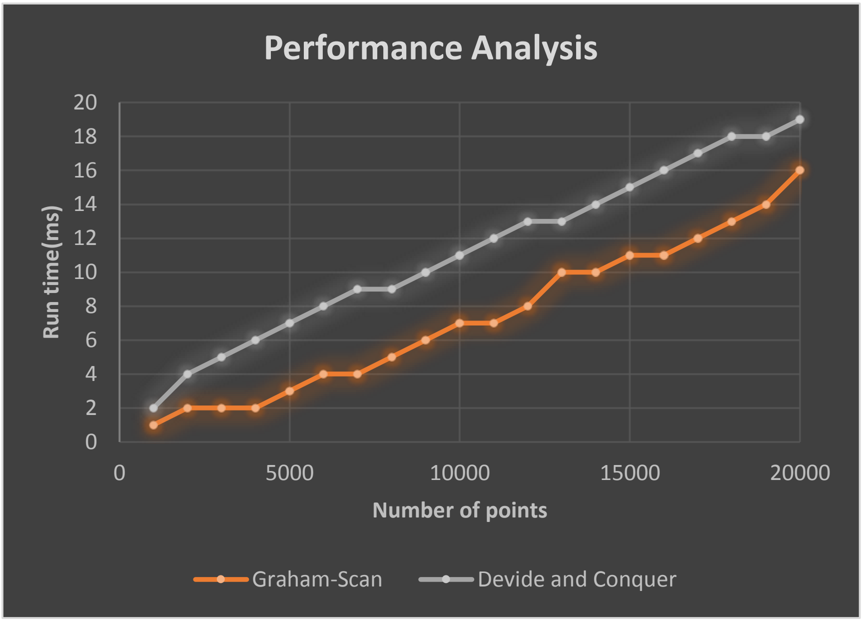


图 5-4 20000 个点以内 Graham-Scan 和分治算法对比

四、实验心得

经过本次实验，我动手实现了三种算法并分析了各自的性能，让我对凸包问题的几种算法有了更加深刻的认知。关于实现方面的心得在上文中已经进行了详尽的叙述。下面主要谈谈自己的感想。

首先，写程序时的小技巧很重要，很可能一个比较笨拙地操作，当执行的次数很多时或数据比较极端的时候，就会带来问题。比如判断点是否在三角形内部时，只需要知道 g(A,B,P)和 g(A,B,C)的乘积的符号，具体乘积是没必要计算的，只需要知道最终的符号就可以了；如果计算，当点坐标比较大时可能会导致溢出。同样，根据 sin 比较极角时，也没必要真的计算 sin 值，经过化简可以避免浮点运算和溢出的危险。

其次，在实现分治算法的过程中，遇到了很多问题和阻碍，因为课件只是提供了一个思路，很多细节需要自己设计。比如分治合并的时候要不要排序的问题，有同学觉得课件上的方法实现起来比较麻烦，就直接手动排序了，但这样就违背了分治法的初衷，导致性能的降低。一开始自己也想放弃，想找到更简便的分治法，但遇到问题解决问题的过程还是很享受的，把难关一个一个克服掉，最终看到分治法成功画出凸包的时候，想想还是有点小激动的。

最后，算法应该是我们的看家本领，是我们安身立命的根本所在，只有学好算法，才能提升内力。完成这次算法实验我的收获很大，对凸包问题也想得很清楚了，以后对待类似的问题，一定要多动手实现，用不同方法实现，最终培养起来用最佳算法解决问题的能力。