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)=Θ(n4)，但如果固定某一个点，遍历其余三个点，时间复杂度降为n3。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。

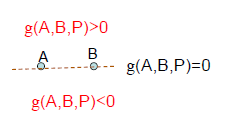


图1-1 点与直线的位置关系

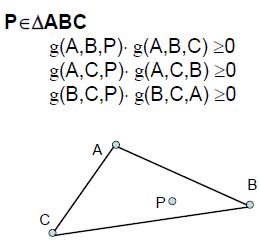


图1-2 判断点在三角形内部

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。

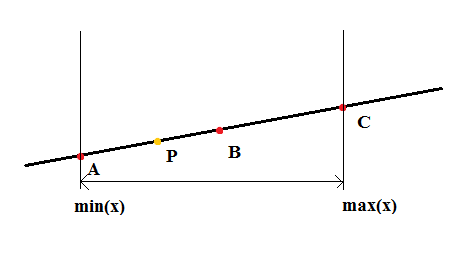


图1-3 共线示意图

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

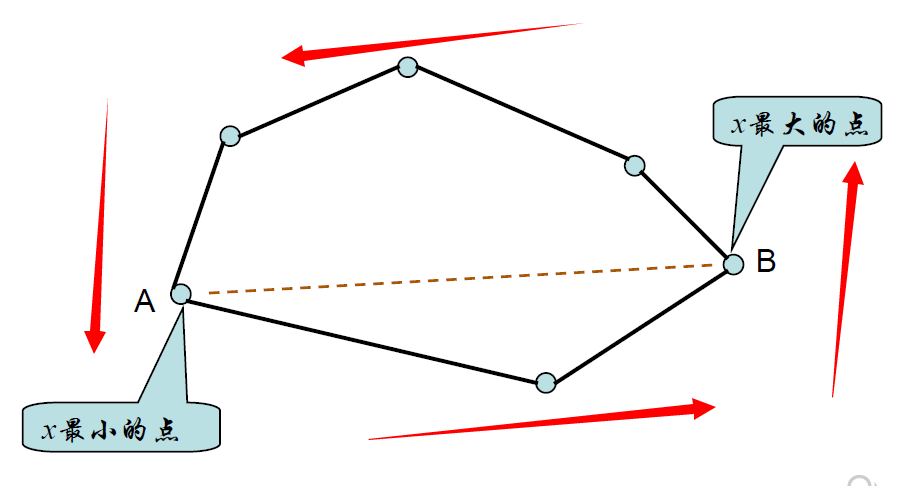


图1-5 合并

综上所示，实现了凸包计算的暴力法，时间复杂度为：

T(n) = Θ(n3)

**（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。

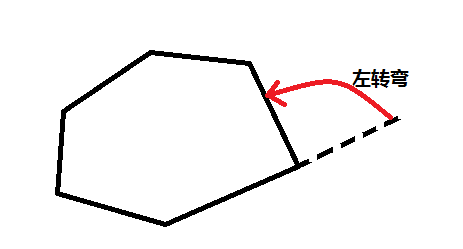


图2-1 左转弯示意图

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);

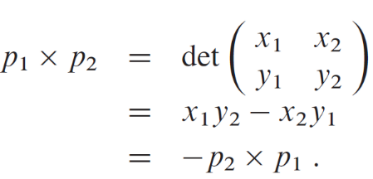
7. Push(pi, S);

8. Rerurn S.

说明如下：

1、Q中y-坐标值最小的点p0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,π)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：



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。

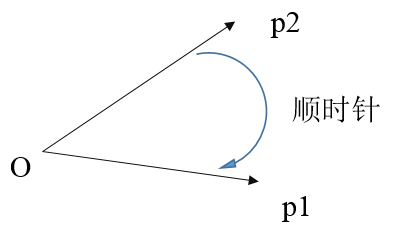


图2-2 叉积示意图

可以使用叉积来对点进行极角排序，因为所有点都在p0上方，故极角范围为(0,π)，在这个范围中，如果某一点和p0组成的向量*p1*在另外一个点与p0组成组成的向量*p2*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。

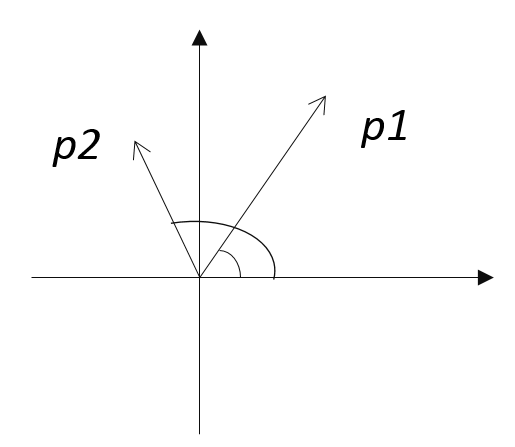


图2-3 使用叉积比较极角

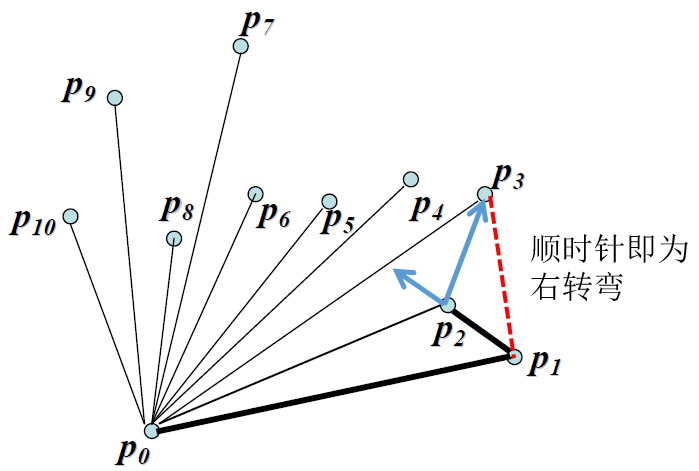


图2-4 使用叉积判断转弯

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QL，QR，左边部分已经返回凸包点集合QL，并且QL中所有点按照逆时针排序，QR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）。

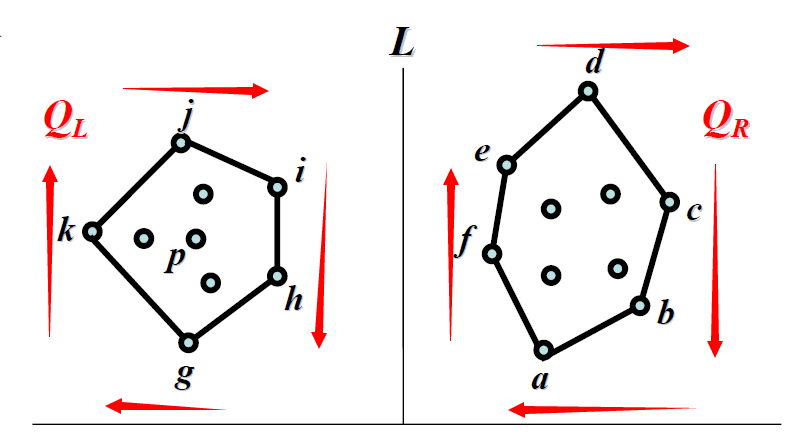


图3-1 合并开始

合并的过程中，首先需要QL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QL中所有点横坐标的平均值，p的纵坐标也为QL中所有点的纵坐标平均值。这样p点肯定在QL内部。下面将对QL，QR中的点相对于p点进行Graham-Scan算法，以找到QL，QR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为：

T(n)=2T(n/2)+O(nlogn)

那么它的时间复杂度肯定会超过Graham-Scan的*n*log*n*，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，QL，QR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QR，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。

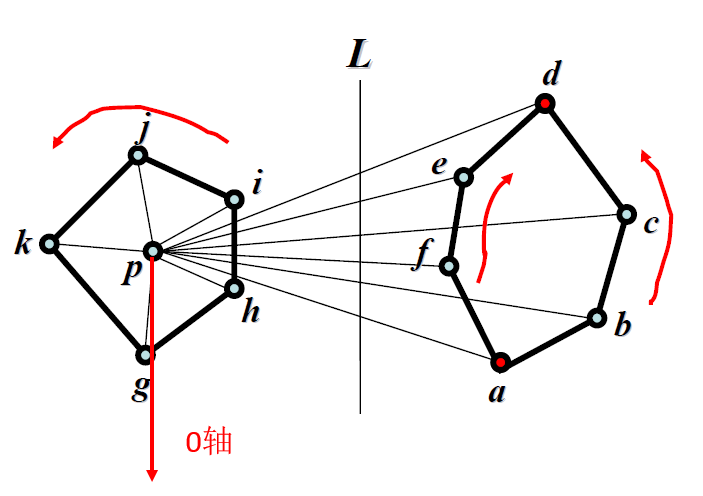


图3-2 极角排序示意

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QL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，QL和QR都是**逆时针**有序的，因此QL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。

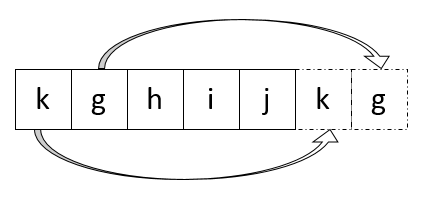


图3-3 顺序调整示意

对于QR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。

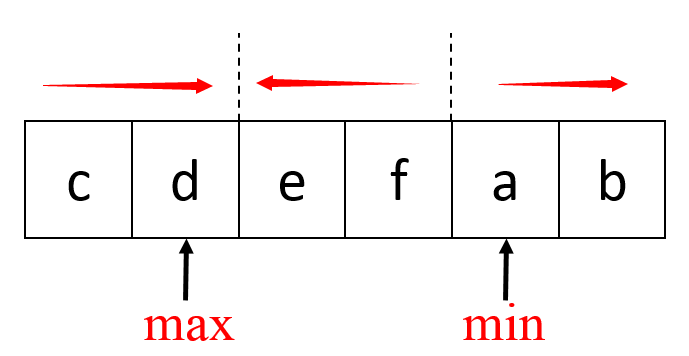


图3-4 QR调整示意

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：

化简为：

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。

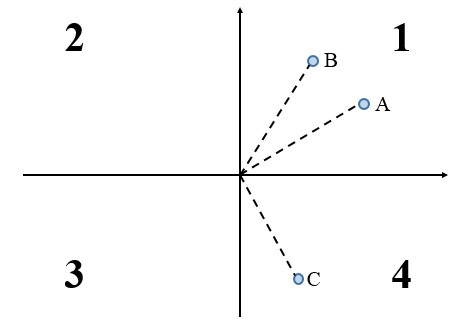


图3-5 极角比较示意

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）。

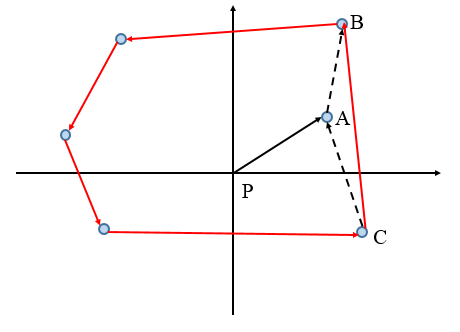


图3-6 扫描结束示意

好了，现在合并过程终于完成了。我们得到的结果是，对于QL和QR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：

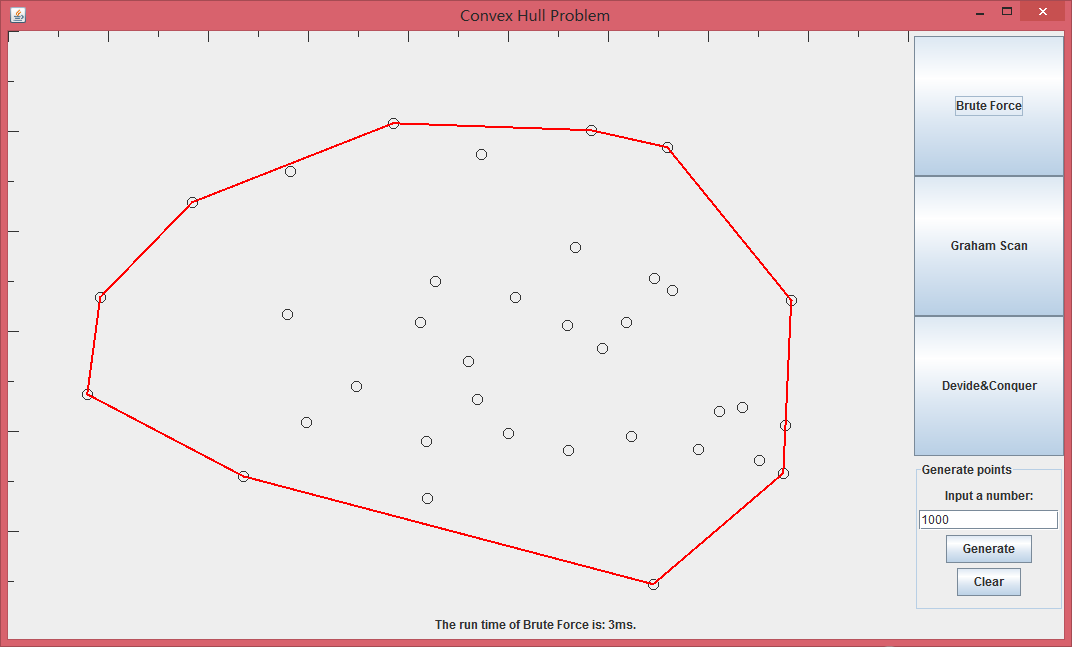


图4-1 程序运行界面

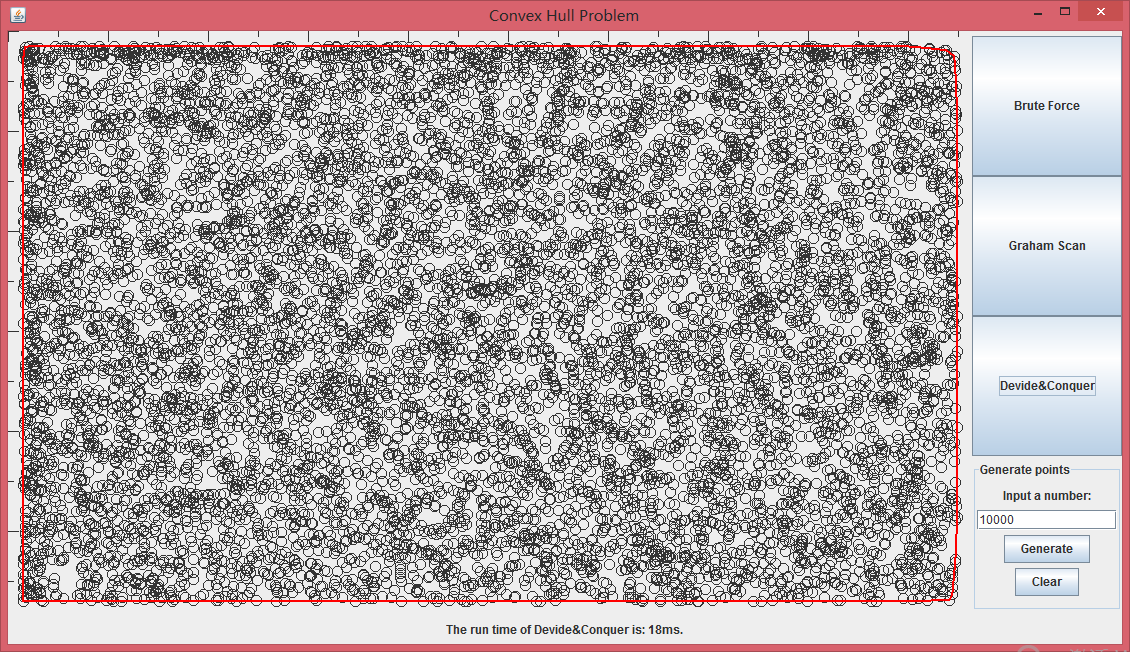


图4-2 10000个点运行界面

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。

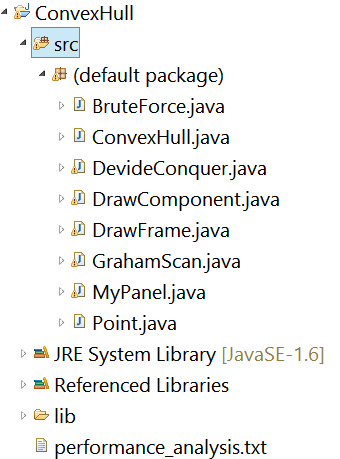
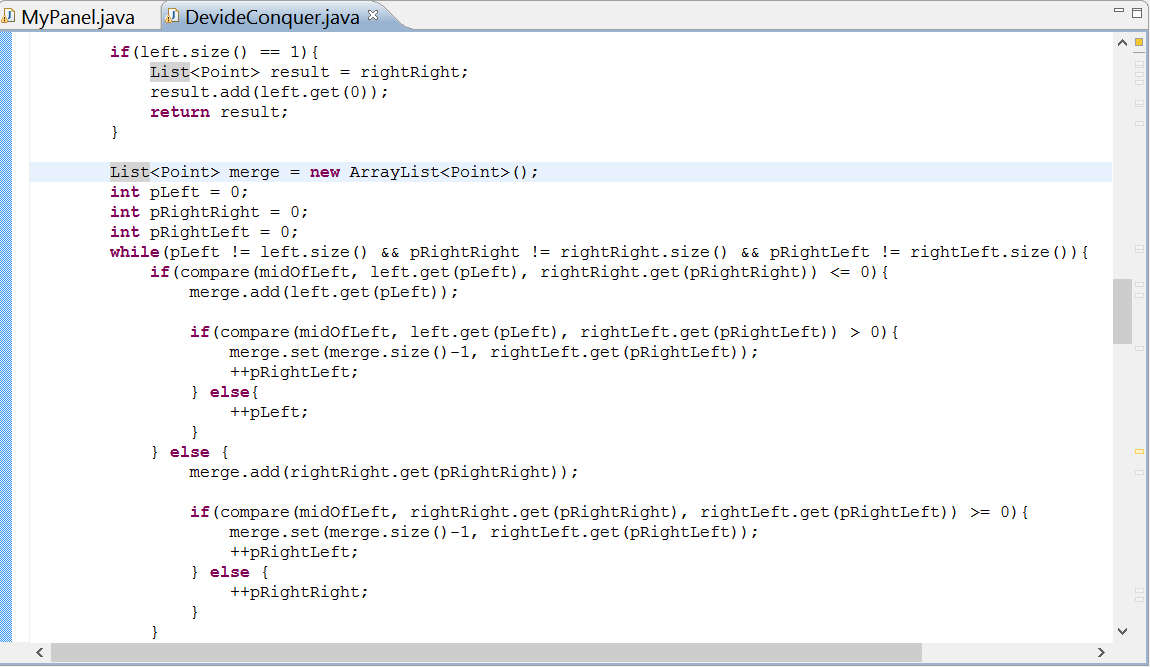


图4-3 工程截图



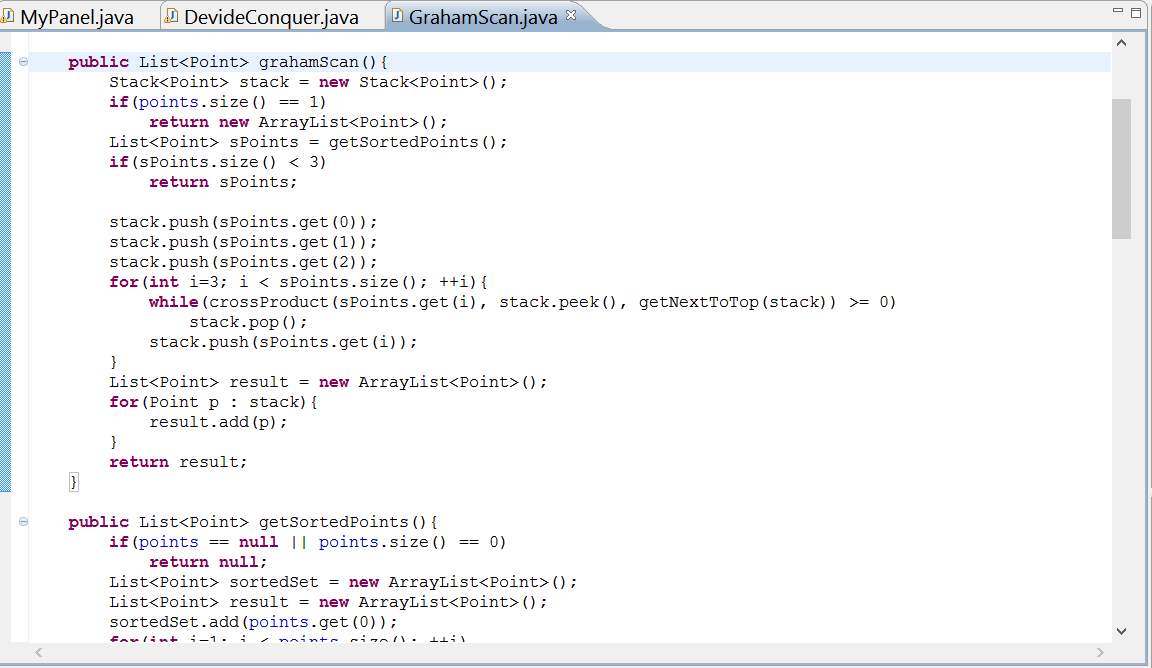


图4-4 部分代码截图

（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。