哈尔滨工业大学

<<算法设计与分析>>

实验报告之一

(2015 年度秋季学期)

姓名：	**成坚**
学号：	**15S003005**
学院：	**计算机学院**
教师：	**高宏**

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

本次实验使用 C 语言完成

1 Brute Force：

```
    for(i = 0; i < n; i++)                  //  循环第1个点
    {
        for(j = i + 1; j < n; j++)          //  循环第2个点
        {
            ///  前两个点组成的直线为ax + by = c 即 ax + by - c = 0;
            a = point[j].y - point[i].y;
            b = point[i].x - point[j].x;
            c = point[i].x * point[j].y - point[i].y * point[j].x;

            left = right = 0;
            for(k = 0; k < n; k++)          //  循环第三个点
            {
                //  将第三个点的坐标带入前两个点连线构成的方程ax + by中
                result = a * point[k].x + b * point[k].y - c;

                if(result < 0)      // ax + by - c < 0点在直线的左侧
                {
                    left++;
                }
                else if(result > 0) //  ax + by -c > 0 点在直线的右侧
                {
                    right++;
                }
```

2 GrahamScan

```
    int pos, leftDownPos;

    leftDownPos = 0;                //  初始位置是第0个位置
    for(pos = 1; pos < n; pos++)    //  找到最左下的点p0
    {
        if((point[pos].y < point[leftDownPos].y)                                //  如果当前点的纵坐标要小与最小点
        || ((point[pos].y == point[leftDownPos].y) && (point[pos].x < point[leftDownPos].x)))     //  或者总左边相同但是横坐标要小
        {
            leftDownPos = pos;
        }
    }
    Swap(&point[0], &point[leftDownPos]);                   // 找到的leftDownPos位置的点

    ///  现在Point中Y坐标最小的点，作为极点，位于Point[0]中
    ///  对顶点按照对极点Point[0]的极角按照从小到大的顺序排序
    ///  对于其中极角相同的点按照离Point[0]的距离从远到近进行排序
    qsort(point + 1, n - 1, sizeof(point[0]), Cmp); //按极角排序


    ///  依次将Point0, Point1, Point2入栈
    for(pos = 0; pos <= 2; pos++)
    {
        stack[pos] = pos; //p0,p1,p2入栈
    }
//    for(top  = 0; top < 3; top++)
//    {
//        stack[top] = top;
//    }
    top = 2;

    for(pos = 3; pos < n; pos++) //最终凸包的各顶点的编号依次存在stack[]中。
    {
        ///  Multi > 0  -=>  表示<Point[strack[top -1]], Point[pos]> 在 <Point[strack[top -1]],point[stack[top]]>的顺时针方向
        ///  也就是说Point[pos]
NORMAL  GrahamScan.c                    GrahamScan() < c                78% ¶ 218:  5    [Syntax: line:100 (1)] ! mixed-indent[249]
```

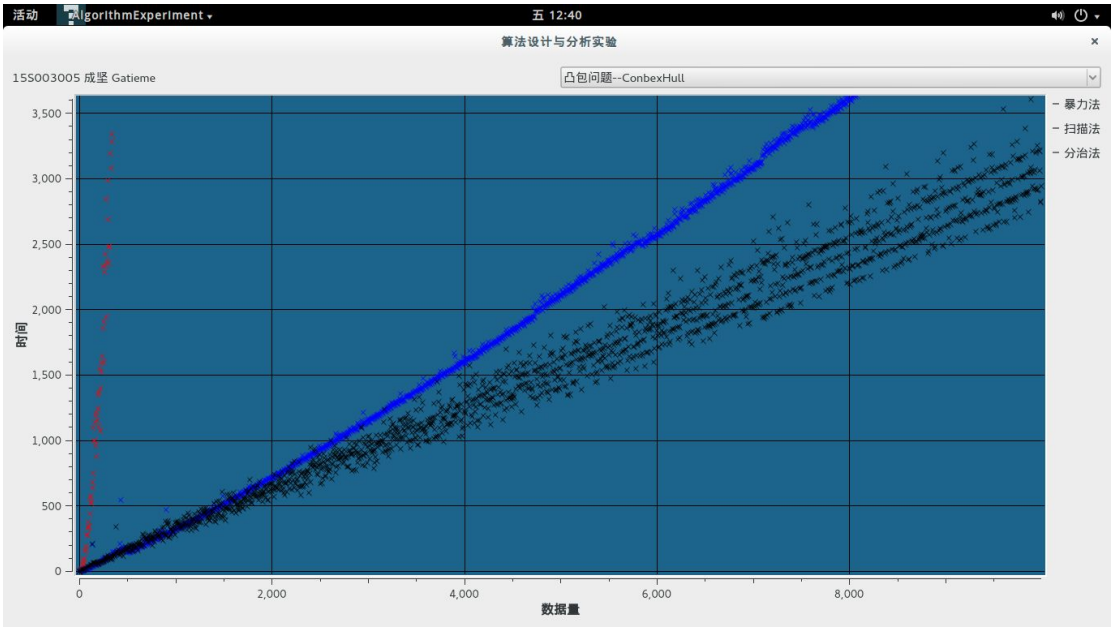
3 Divide

```
void DealWithLeft(int first, int final, Point *point, Point *convex)
{
    int max = 0, index = -1;
    int i = first;
    if(first < final)                   //   point[first]->point[final]射线左侧
    {
        for(; i < final; i++)
        {
            int x1 = point[first].x,y1 = point[first].y;
            int x2 = point[final].x,y2 = point[final].y;
            int x3 = point[i].x;
            int y3 = point[i].y;


            int compute = x1 * y2 + x3 * y1 + x2 * y3 - x3 * y2 - x2 * y1 - x1 * y3;
            if(compute > max)
            {
                max = compute;
                index = i;
            }
        }
    }
    else
    {
        for(; i >= 0; i--)//point[final]->point[first]射线左侧
        {
            int x1 = point[first].x,y1 = point[first].y;
            int x2 = point[final].x,y2 = point[final].y;
            int x3 = point[i].x,y3 = point[i].y;


            int compute = x1 * y2 + x3 * y1 + x2 * y3 - x3 * y2 - x2 * y1 - x1 * y3;
            if(compute > max)
            {
                max = compute;
NORMAL  DivideConvex.c                  DealWithLeft() < c  81% ¶ 197: 17  [Syntax: line:80 (1)] ! mixed-indent[66]
```

4 绘图曲线



对比可以发现，BruteForce 算法的增长最快，并且性能很不稳定，是$O(n^4)$的算法，但是已经处理过的点不再处理，所以会有不稳定因素出现，也实际上达不到n^4级别的消耗。另外两种算法的增长有同样的分布，但是分治法相对较为耗时。当算法输入点为整数点集时，分治算法在 10W 点以上会有相对较好的性能(图中是浮点数情况)，而浮点情况下 grahams 扫描更好一些。两种算法都是$O(n\log n)$的。

四、实验心得

本次实验中 graham 实现最为容易，因为毕竟是现有的已证明的好算法。

BruteForce 中有一些边界细节需要考虑，分治有很多细节要考虑，最主要的情况就是输入全为一条线上的点（对分治来说，即使输入不是这样的，数据量大时仍然很有可能分出一个全部在一条竖线上的子问题出来）。

分治还有个问题就是所谓的“QL”内点，如果真的按内点的话，那合并时的 graham 扫描显然还要再找一次最低点，需要改造算法，不如直接每次取左凸包的左上角点。