

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 :

```

25     for(i = 0; i < n; i++)           // 循环第1个点
24     {
23         for(j = i + 1; j < n; j++)   // 循环第2个点
22         {
21             // 前两个点组成的直线为  $ax + by = c$  即  $ax + by - c = 0$ ;
20             a = point[j].y - point[i].y;
19             b = point[i].x - point[j].x;
18             c = point[i].x * point[j].y - point[i].y * point[j].x;
17
16             left = right = 0;
15             for(k = 0; k < n; k++)   // 循环第三个点
14             {
13                 // 将第三个点的坐标带入前两个点连线构成的方程  $ax + by = c$ 
12                 result = a * point[k].x + b * point[k].y - c;
11
10                 if(result < 0)       //  $ax + by - c < 0$  点在直线的左侧
9                 {
8                     left++;
7                 }
6                 else if(result > 0) //  $ax + by - c > 0$  点在直线的右侧
5                 {
4                     right++;
3                 }
2             }

```

2 GrahamScan

```

26     int pos, leftDownPos;
25
24     leftDownPos = 0;           // 初始位置是第0个位置
23     for(pos = 1; pos < n; pos++) // 找到最左下的点p0
22     {
21         if((point[pos].y < point[leftDownPos].y)
20         || ((point[pos].y == point[leftDownPos].y) && (point[pos].x < point[leftDownPos].x))) // 如果当前点的纵坐标要小与最小点
19         { // 或者总左边相同但是横坐标要小
18             leftDownPos = pos;
17         }
16     }
15     Swap(&point[0], &point[leftDownPos]); // 找到的leftDownPos位置的点
14
13     // 现在point中y坐标最小的点, 作为极点, 位于point[0]中
12     // 对顶点按照对极点point[0]的极角按照从小到大的顺序排序
11     // 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
10     qsort(point + 1, n - 1, sizeof(point[0]), Cmp); // 按极角排序
9
8     // 依次将point0, point1, point2入栈
7     for(pos = 0; pos <= 2; pos++)
6     {
5         stack[pos] = pos; // p0, p1, p2入栈
4     }
3     for(top = 0; top < 3; top++)
2     {
1         stack[top] = top;
218 //
1 //
2     top = 2;
3     for(pos = 3; pos < n; pos++) // 最终凸包的各顶点的编号依次存在stack[]中。
4     {
5         // Multit > 0 -> 表示<Point[stack[top - 1]], Point[pos]> 在 <Point[stack[top - 1]], point[stack[top]]>的顺时针方向
6         // 也就是说point[pos]
7         //

```

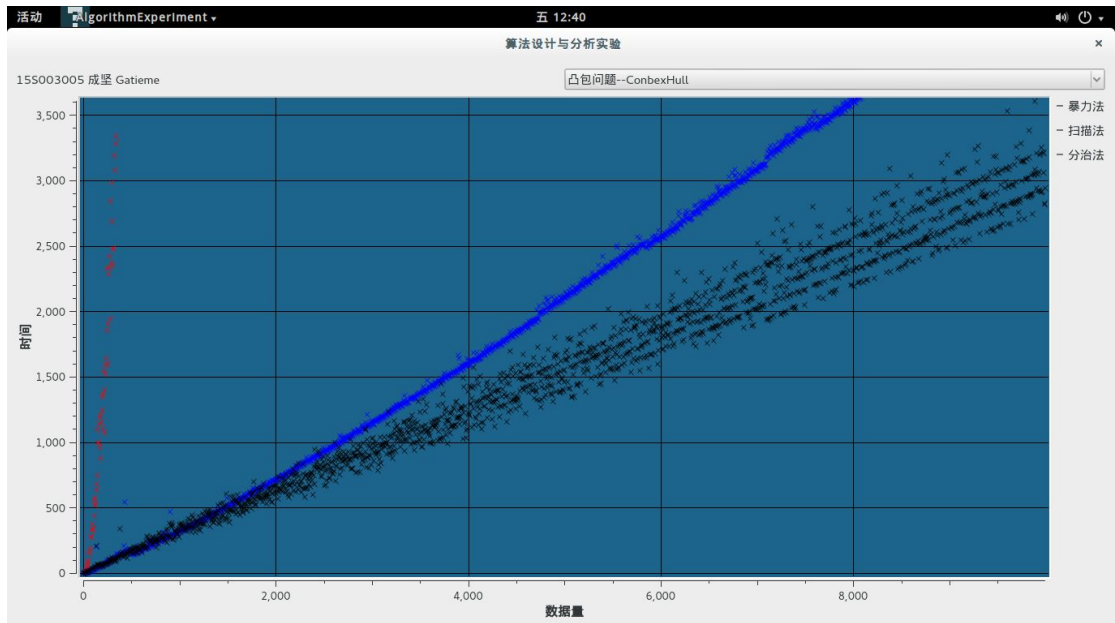
3 Divide

```

17 void DealWithLeft(int first, int final, Point *point, Point *convex)
16 {
15     int max = 0, lindex = -1;
14     int l = first;
13     if(first < final)           // point[first]->point[final]射线左侧
12     {
11         for(; l < final; l++)
10         {
9             int x1 = point[first].x, y1 = point[first].y;
8             int x2 = point[final].x, y2 = point[final].y;
7             int x3 = point[l].x;
6             int y3 = point[l].y;
5
4             int compute = x1 * y2 + x3 * y1 + x2 * y3 - x3 * y2 - x2 * y1 - x1 * y3;
3             if(compute > max)
2             {
197                 max = compute;
198                 lindex = l;
199             }
198         }
197     }
196     else
195     {
194         for(; l >= 0; l--) // point[final]->point[first]射线左侧
193         {
192             int x1 = point[first].x, y1 = point[first].y;
191             int x2 = point[final].x, y2 = point[final].y;
190             int x3 = point[l].x, y3 = point[l].y;
189
188             int compute = x1 * y2 + x3 * y1 + x2 * y3 - x3 * y2 - x2 * y1 - x1 * y3;
187             if(compute > max)
186             {
185                 max = compute;
184                 lindex = l;
183             }
182         }
181     }
180 }
179
178
177
176
175
174
173
172
171
170
169
168
167
166
165
164
163
162
161
160
159
158
157
156
155
154
153
152
151
150
149
148
147
146
145
144
143
142
141
140
139
138
137
136
135
134
133
132
131
130
129
128
127
126
125
124
123
122
121
120
119
118
117
116
115
114
113
112
111
110
109
108
107
106
105
104
103
102
101
100
99
98
97
96
95
94
93
92
91
90
89
88
87
86
85
84
83
82
81
80
79
78
77
76
75
74
73
72
71
70
69
68
67
66
65
64
63
62
61
60
59
58
57
56
55
54
53
52
51
50
49
48
47
46
45
44
43
42
41
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

```

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 graham's 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。