

## 数据结构与算法 II 上机实验 (9.25)

中国人民大学 信息学院 崔冠宇 2018202147

**上机题** 用分治策略找出平面点集中最接近点对。

**解：**由于代码过长，附在最后，此处仅给出算法框架（伪代码）：

---

**Algorithm** ( $set, dis$ ) =  $nearest\_pair(X, Y)$  (分治法解决平面最接近点对问题)

---

**输入：**按  $x$  分量从小到大排序的平面点集  $X$ 、按  $y$  分量从小到大排序的平面点集  $Y$ 。

**输出：**最近点对集合，最近点对之间的距离。

**边界条件：** $|X| = |Y| \leq 3$  时，直接求出最近点对集合和最近距离，返回。

1. 求出点集  $x$  分量的中位数  $median$ 。
2. 用  $x = median$  分别划分  $X$  和  $Y$  为  $XL, XR, YL, YR$  为左右两大部分（保持有序）。
3. **递归调用：**  
 $(lset, ldis) = nearest\_pair(XL, YL)$ ,  
 $(rset, rdis) = nearest\_pair(XR, YR)$ 。
4.  $dis = \min\{ldis, rdis\}$ ，同时根据左右大小设置  $set$  初值。
5. 将  $Y$  中  $x$  分量属于  $[median - dis, median + dis]$  的点取出，构成  $Y'$ （保持有序）。
6. 对  $Y'$  中每个点，检查与其后至多 7 个点的距离<sup>a</sup>，并根据情况更新  $set$  和  $dis$ 。
7. 返回  $set$  和  $dis$ 。

---

<sup>a</sup>根据课本上的证明。

**算法 (时间) 复杂度分析：**

1. 预排序部分：在递归之外， $O(n \log n)$ ；
2. 中位数部分：由于  $X$  是排好序的，直接根据长度可算出中位数对应的下标，故为  $O(1)$ ；
3. 划分部分：只需要扫描两个数组即可，故为  $O(n)$ ；
4. 递归部分：划分为两个规模相同的子问题，故为  $2T(n/2)$ ；
5. 取  $Y'$ ：扫描  $Y$  即可，故为  $O(n)$ ；
6. 检查跨界情况：由于对每个点至多检查 7 对距离，故为  $O(n)$ 。

可得递推式  $T(n) = 2T(n/2) + O(n)$ ，结合边界条件是  $O(1)$  的，以及  $\mathbb{T}(n) = T(n) + O(n \log n)$ ，根据主定理，

$$\mathbb{T}(n) = T(n) = O(n \log n).$$

程序运行截图：

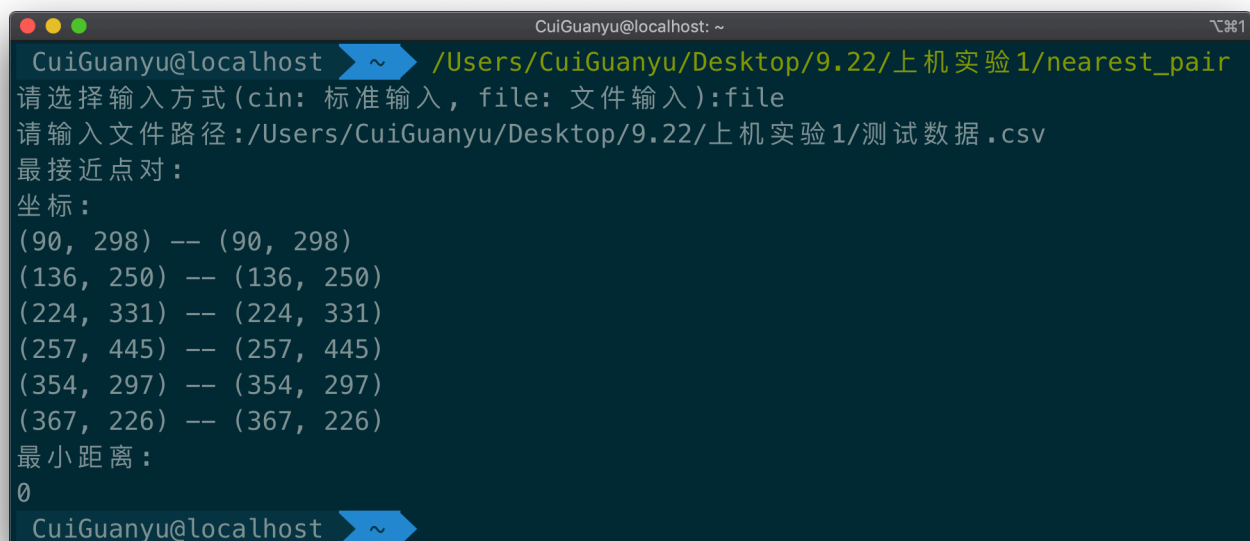
我实现了两种输入方式，一种是对较小规模数据手动输入，另一种是从 csv 文件读取（需要将老师给的 excel 文件另存为 csv 文件）。以下是两种方式的运行截图：

**std::cin 读入方式：**

A terminal window titled 'CuiGuanyu@localhost: ~' showing the execution of a program. The user enters 'cin' to select input method, then '5' for the number of points, and then coordinates '0 0 1 1 2 2 3 3 4 4'. The program outputs the closest point pairs and the minimum distance.

```
CuiGuanyu@localhost ~ /Users/CuiGuanyu/Desktop/9.22/上机实验1/nearest_pair
请选择输入方式(cin: 标准输入, file: 文件输入):cin
请输入点总数(正整数):5
请输入各点坐标:
(格式: x y)
0 0 1 1 2 2 3 3 4 4
最接近点对:
坐标:
(0, 0) -- (1, 1)
(1, 1) -- (2, 2)
(2, 2) -- (3, 3)
(3, 3) -- (4, 4)
最小距离:
1.41421
CuiGuanyu@localhost ~
```

**std::fstream 读入方式：**

A terminal window titled 'CuiGuanyu@localhost: ~' showing the execution of a program. The user enters 'file' to select input method, then the file path '/Users/CuiGuanyu/Desktop/9.22/上机实验1/测试数据.csv'. The program outputs the closest point pairs and the minimum distance, which is 0.

```
CuiGuanyu@localhost ~ /Users/CuiGuanyu/Desktop/9.22/上机实验1/nearest_pair
请选择输入方式(cin: 标准输入, file: 文件输入):file
请输入文件路径:/Users/CuiGuanyu/Desktop/9.22/上机实验1/测试数据.csv
最接近点对:
坐标:
(90, 298) -- (90, 298)
(136, 250) -- (136, 250)
(224, 331) -- (224, 331)
(257, 445) -- (257, 445)
(354, 297) -- (354, 297)
(367, 226) -- (367, 226)
最小距离:
0
CuiGuanyu@localhost ~
```

附录: **nearest\_pair.cpp**:

```
1 // -----
2 //             平面最接近点对
3 //             崔冠宇 2020.09.24
4 // -----
5
6 // std::cin & std::cout
7 #include <iostream>
8 // std::fstream
9 #include <fstream>
10 // std::vector<T>
11 #include <vector>
12 // std::string
13 #include <string>
14 // std::set<T>
15 #include <set>
16 // std::pair<T1, T2>
17 #include <utility>
18 // std::numeric_limits<T>::infinity()
19 #include <limits>
20 // std::sqrt()
21 #include <cmath>
22
23 // 定义平面上点的类型为 double 数对
24 typedef std::pair<double, double> Point2D;
25
26 // 传入按X分量排好的点的序列()、按Y分量排好的序列
27 // (点集P用P_X也可表示, 节省空间)
28 // 返回最近点对的std::set及最近距离
29 // 返回类型: std::pair<std::set<std::pair<Point2D, Point2D>>, double>
30 auto nearestPair(const std::vector<Point2D> & P_X, const std::vector<Point2D> & Y)
31 {
32     // 点对的 std::vector
33     std::set<std::pair<Point2D, Point2D>> nearest;
34     // 最短距离的平方设为无穷大
35     double minDisSquare = std::numeric_limits<double>::infinity();
36     if(P_X.size() <= 1)
```

```

37     {
38         return std::pair(nearest, minDisSquare);
39     }
40     // n 较小的情况, O(1)
41     if(P_X.size() <= 3)
42     {
43         // 两两计算点对之间的距离
44         for(size_t i = 0; i < P_X.size(); i++)
45         {
46             for(size_t j = i + 1; j < P_X.size(); j++)
47             {
48                 double dx = P_X[i].first - P_X[j].first;
49                 double dy = P_X[i].second - P_X[j].second;
50                 double disSquare = dx * dx + dy * dy;
51                 // 若比之前最小值还小, 更新
52                 if(disSquare < minDisSquare || minDisSquare - disSquare > 1e-6)
53                 {
54                     minDisSquare = disSquare;
55                     // 清空原来的最近点对集合
56                     nearest.clear();
57                     nearest.insert(std::pair<Point2D, Point2D>(P_X[i], P_X[j]));
58                 }
59                 // 相同距离点对, 直接插入集合
60                 else if(std::abs(disSquare - minDisSquare) < 1e-6)
61                 {
62                     nearest.insert(std::pair<Point2D, Point2D>(P_X[i], P_X[j]));
63                 }
64             }
65         }
66         // 返回
67         return std::pair(nearest, std::sqrt(minDisSquare));
68     }
69     // 分解为子问题
70     // 计算中位数
71     size_t size = P_X.size();
72     size_t median = (size % 2 == 0) ? (size / 2 - 1) : (size / 2);
73

```

```

74 // 划分问题的点集为 x in [0, median], (median, size - 1] 左右两半部分
75 std::vector<Point2D> PL_XL, PR_XR, YL, YR;
76 // 按在左边还是右边划分P_X
77 // 可以直接按中位数左右划分
78 // T(n)=O(n)
79 for(size_t i = 0; i <= median; i++)
80 {
81     PL_XL.push_back(P_X[i]);
82 }
83 for(size_t i = median + 1; i < P_X.size(); i++)
84 {
85     PR_XR.push_back(P_X[i]);
86 }
87 // 按x分量在左边还是右边划分Y
88 // T(n)=O(n)
89 for(size_t i = 0; i < Y.size(); i++)
90 {
91     if(Y[i].first <= P_X[median].first)
92     {
93         YL.push_back(Y[i]);
94     }
95     else
96     {
97         YR.push_back(Y[i]);
98     }
99 }
100 // 递归地调用两边的子问题
101 // 2T(n/2)
102 auto ansLeft = nearestPair(PL_XL, YL);
103 auto ansRight = nearestPair(PR_XR, YR);
104 // 左右的最短距离
105 double dleft = ansLeft.second;
106 double dright = ansRight.second;
107 // 取左右较小的作为d
108 double dminLeftRight;
109 // 指示取的是左边还是右边的结果
110 bool leftmin = false;

```

```
111 // 指示左右是否相等
112 bool equal = false;
113 // 左边距离较小
114 if(dleft < dright)
115 {
116     dminLeftRight = dleft;
117     leftmin = true;
118 }
119 // 右边距离较小
120 else if(dleft > dright)
121 {
122     dminLeftRight = dright;
123 }
124 // 两边相等
125 else
126 {
127     equal = true;
128     dminLeftRight = dleft;
129 }
130
131 // 下面处理跨线的情况
132 // 最终返回的的最小距离点对
133 std::set<std::pair<Point2D, Point2D>> ret;
134 // 集合复制必然是 O(n) 的，根据C++标准甚至更低一些
135 // 左边较小，取左边的所有最近点对的集合
136 if(leftmin)
137 {
138     ret = ansLeft.first;
139 }
140 // 两边相等，取并集
141 else if(equal)
142 {
143     ret = ansLeft.first;
144     for(auto i = ansRight.first.begin(); i != ansRight.first.end(); i++)
145     {
146         ret.insert(*i);
147     }
```

```

148     }
149     // 右边较小，取右边
150     else
151     {
152         ret = ansRight.first;
153     }
154     // 先对Y进行处理，选择在[m-d, m+d]条带内的点
155     std::vector<Point2D> YStrip;
156     // T(n)=O(n)
157     for(size_t i = 0; i < Y.size(); i++)
158     {
159         if(Y[i].first >= P_X[median].first - dminLeftRight
160             && Y[i].first <= P_X[median].first + dminLeftRight)
161         {
162             YStrip.push_back(Y[i]);
163         }
164     }
165     // 根据课本上的证明，对于条带中的每个点，只需要检查后面至多七个点
166     // T(n)=O(n)
167     for(size_t i = 0; i < YStrip.size(); i++)
168     {
169         for(size_t j = 1; j <= 7 && i + j < YStrip.size(); j++)
170         {
171             double dx = YStrip[i].first - YStrip[i + j].first;
172             double dy = YStrip[i].second - YStrip[i + j].second;
173             double disSquare = dx * dx + dy * dy;
174             // 如果得到跨界更小的，更新最小距离点对集合
175             if(disSquare < dminLeftRight * dminLeftRight - 1e-3
176                 || dminLeftRight * dminLeftRight - disSquare > 1e-3)
177             {
178                 dminLeftRight = std::sqrt(disSquare);
179                 ret.clear();
180                 ret.insert(std::pair(YStrip[i], YStrip[i + j]));
181             }
182             // 相同距离点对
183             else if(std::abs(disSquare - dminLeftRight * dminLeftRight)
184                 < 1e-3)

```

```

185         {
186             ret.insert(std::pair(YStrip[i], YStrip[i + j]));
187         }
188     }
189 }
190 //  $T(n) = 2T(n/2) + O(n)$ 
191 // -->  $T(n) = O(n \log n)$ 
192 return std::pair(ret, dminLeftRight);
193 }
194
195 int main(int argc, char *argv[])
196 {
197     // 《算法导论》P611-613
198     // 按x分量排序的数组 -- X
199     std::vector<Point2D> pointsX;
200     // 按y分量排序的数组 -- Y
201     std::vector<Point2D> pointsY;
202     // 选择输入方式——手动or读文件
203     std::cout << "请选择输入方式(cin: 标准输入, file: 文件输入):";
204     std::string command;
205     std::cin >> command;
206     if(command == "cin")
207     {
208         // 点总数
209         size_t totalCount = 0;
210         // 输入点总数
211         std::cout << "请输入点总数(正整数):";
212         std::cin >> totalCount;
213         // 输入各点对
214         std::cout << "请输入各点坐标:"
215             << std::endl << "(格式: x y)"
216             << std::endl;
217         double x = 0, y = 0;
218         // 以下对于复杂度的讨论, 问题规模(n)指点的数目.
219         // 输入过程:  $T(n)=O(n)$ 
220         for(size_t i = 0; i < totalCount; i++)
221         {

```



```
222         std::cin >> x >> y;
223         pointsX.push_back(Point2D(x, y));
224         pointsY.push_back(Point2D(x, y));
225     }
226 }
227 else if(command == "file")
228 {
229     std::string filepath;
230     std::cout << "请输入文件路径:";
231     std::cin >> filepath;
232     std::fstream f(filepath);
233     double x = 0, y = 0;
234     char comma = ',';
235     while(!f.eof())
236     {
237         f >> x >> comma >> y;
238         pointsX.push_back(Point2D(x, y));
239         pointsY.push_back(Point2D(x, y));
240     }
241 }
242 else
243 {
244     std::cout << "不合法输入。" << std::endl;
245     return 0;
246 }
247
248 // 排序用，lambda表达式比较两点大小
249 auto CompareX = [](Point2D a, Point2D b) -> bool
250 {
251     // x 分量从小到大
252     return (a.first < b.first
253     // x 分量相同，y 分量从小到大
254         || (a.first == b.first && a.second < b.second));
255 };
256 auto CompareY = [](Point2D a, Point2D b) -> bool
257 {
258     // x 分量从小到大
```

```

259         return (a.second < b.second
260             // x 分量相同, y 分量从小到大
261             || (a.second == b.second && a.first < b.first));
262     };
263     // 预排序过程:  $T(n)=O(n \log n)$ 
264     std::sort(pointsX.begin(), pointsX.end(), CompareX);
265     std::sort(pointsY.begin(), pointsY.end(), CompareY);
266     // 调用
267     auto res = nearestPair(pointsX, pointsY);
268     // 输出
269     if(res.first.size() == 0)
270     {
271         std::cout << "无最接近点对。" << std::endl;
272         return 0;
273     }
274     std::cout << "最接近点对:"
275         << std::endl << "坐标:" << std::endl;
276
277     for(auto i = res.first.begin(); i != res.first.end(); i++)
278     {
279         std::cout << "(" << (*i).first.first << ", "
280         << (*i).first.second << ") -- ("
281         << (*i).second.first << ", "
282         << (*i).second.second << ")"
283         << std::endl;
284     }
285     std::cout << "最小距离:"
286         << std::endl << res.second << std::endl;
287     return 0;
288 }

```