## 最近点对实验报告

操作系统：win10

编程语言：python3.6

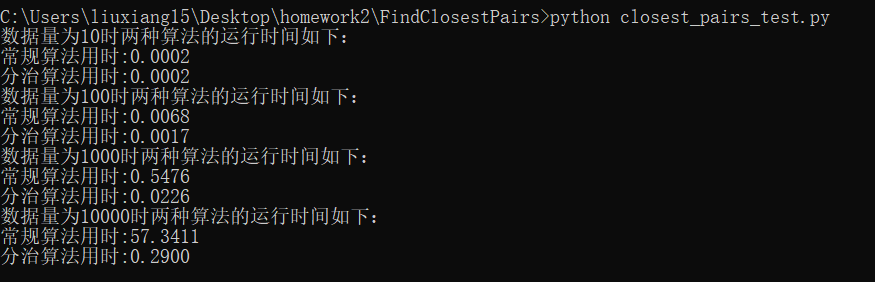
依赖包：numpy,random,math, operator,time,copy

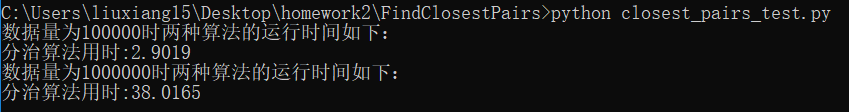
运行说明：

如果想比较分治算法和暴力法求最近点对的运行时间，请在closest\_pairs\_test.py所在目录运行python closest\_pairs\_test.py即可

如果想使用GUI并点击加入点查看最短距离和点对信息，请在closest\_pairs\_UI.py所在目录运行python closest\_pairs\_UI.py即可

运行结果





在不同数据量下两种算法运行的时间

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 100 |  |  |  |  |
| 常规算法(O(n^2)) | 0.0002 | 0.0068 | 0.5476 | 57.3411 | 时间过长 | 时间过长 |
| 分治算法  (O(nlogn)) | 0.0002 | 0.0017 | 0.0226 | 0.2900 | 2.9019 | 380165 |

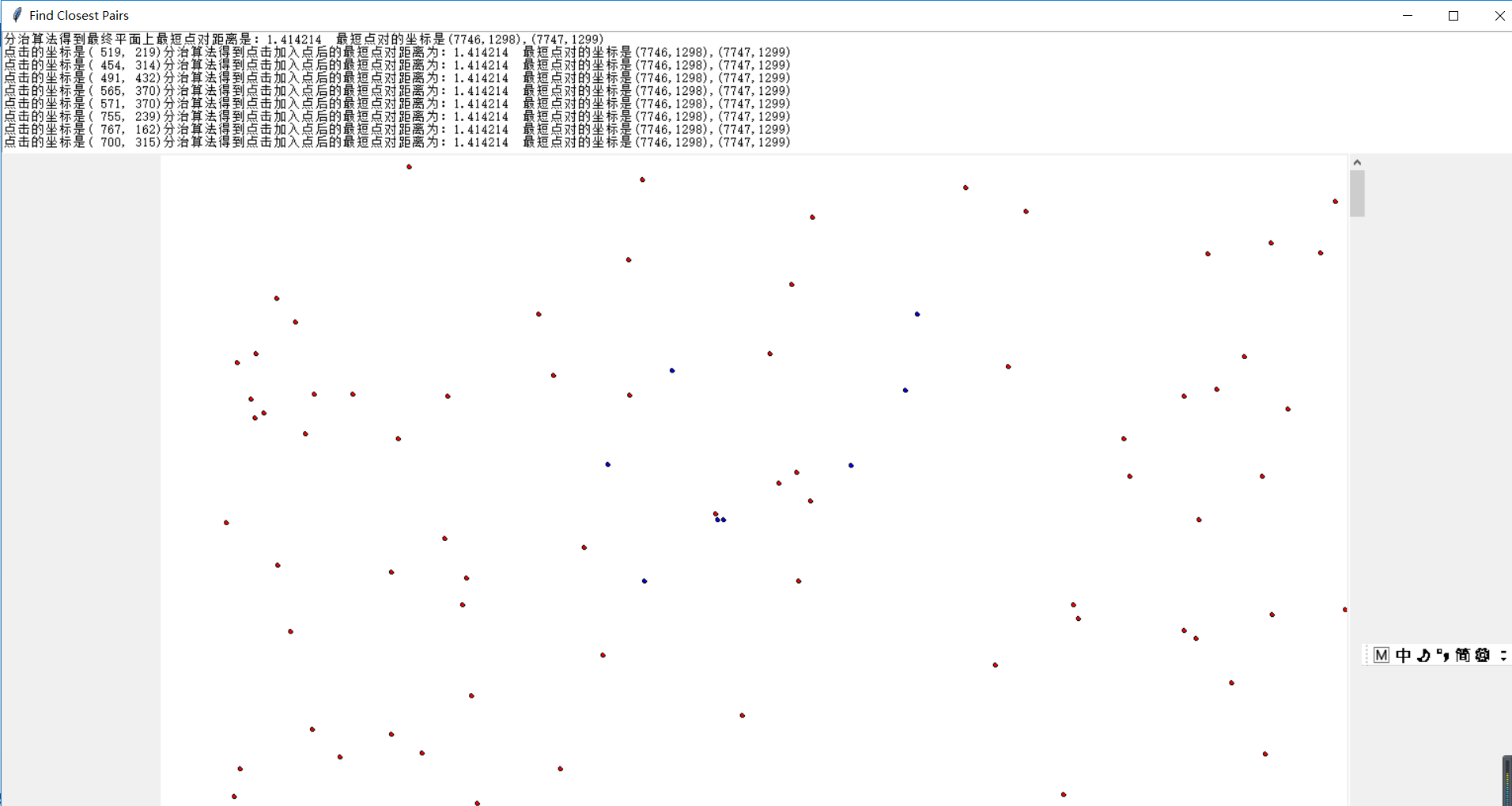
由运行时间比较可知，随着数据量增长10倍，

常规算法运行时间大致增长为100倍。

分治算法运行时间大致增长倍数在[8,20]之间

总的来说时间复杂度上大致符合。

### 图形界面



## 方法分析

### 预处理：

生成随机点：使用numpy模块的random.random\_integers()函数分别生成在平面内的随机点的x,y坐标(可能会包含重复点)

分治算法预处理：对所有点根据x坐标进行排序，返回新的点数组

### 分治算法：

1. 递归边界设定

如果数组中只有1个元素，返回MAX\_DISTANCE(+∞)

如果数组中只有2个元素，返回两点的距离

def get\_closest\_pairs(self, left, right):

if left == right:

return MAX\_DISTANCE

if left+1 == right:

dis = self.get\_distance(self.points\_list[left], self.points\_list[right])

if dis < self.curr\_distance:

self.curr\_distance = dis

self.point\_a = self.points\_list[left]

self.point\_b = self.points\_list[right]

return dis

1. 将所有平面上的点根据x坐标分成左右两部分，递归分别求左右两部分的最近点对距离，取较小值

mid = (left + right) // 2 #取x的中点

d1 = self.get\_closest\_pairs(left, mid)

d2 = self.get\_closest\_pairs(mid, right)

d = min(d1, d2)

1. 将离中心点points\_list[mid]的横向(x)距离不小于d的点加入缓存数组temp\_list并按照y坐标排序

temp\_list=[]

i = mid-1

while i >= 0 and (self.points\_list[mid]['x']-self.points\_list[i]['x']) <= d:

temp\_list.append(self.points\_list[i])

i -= 1

j = mid+1

while j <= right and (self.points\_list[j]['x']-self.points\_list[mid]['x']) <= d:

temp\_list.append(self.points\_list[i])

j += 1

temp\_list = self.sort\_y(temp\_list)

1. 对缓存数组temp\_list中的每一个元素point遍历,

遍历point之后的点point’

如果point’与point的纵向(y)距离不大于d的话，

计算point与point’之间的距离，

如果小于d则更新最短距离和最短点对

否则退出本层循环

否则point循环更新

temp\_len = len(temp\_list)

for i in range(temp\_len):

for j in range(i+1, temp\_len):

#不超过六个点进入此循环

if temp\_list[j]['y'] - temp\_list[i]['y'] < min\_dis:

d3 = self.get\_distance(temp\_list[i], temp\_list[j])

min\_dis = min(min\_dis, d3)

#加入寻找最近点对的相关代码

if min\_dis == d3 and min\_dis < self.curr\_distance:

self.curr\_distance = min\_dis

self.point\_a = temp\_list[i]

self.point\_b = temp\_list[j]

else:

break

return min\_dis

### 暴力算法

设定初始距离为无穷大，分别求两点之间距离，如果小于初始距离更新最短距离和最短点对。

def brute\_closest\_pairs(self, points\_list):

init\_distance = MAX\_DISTANCE

for i in range(self.size):

for j in range(i+1, self.size):

dis = self.get\_distance(points\_list[i], points\_list[j])

if dis < init\_distance:

init\_distance = dis

self.point\_a = points\_list[i]

self.point\_b = points\_list[j]

### GUI设计：

引入tkinter做GUI

正如前图，界面功能如下：

* 最上的文本框显示分治算法求得最短点对和距离信息。
* 点击界面会在平面上加入绿色的点并重新计算最短点对和距离信息并显示在文本框
* 点击滚动框可以移动界面