**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与分析**

**实验名称： 分治法求最近点对问题**

**学院： 计算机与软件学院 专业： 软件工程**

**报告人： 蔡晓鑫 学号： 2017151019 班级： 01**

**同组人： 无**

**指导教师： 杨煊**

**实验时间： 2019/3/22——2019/4/15**

**实验报告提交时间： 2019/4/10**

**教务处制**

**一、实验目的：**

（1） 掌握分治法思想。

（2） 学会最近点对问题求解方法。

**二、内容：**

1. 对于平面上给定的N个点，给出所有点对的最短距离，即，输入是平面上的N个点，输出是N点中具有最短距离的两点。

2. 要求随机生成N个点的平面坐标，应用蛮力法编程计算出所有点对的最短距离。

3. 要求随机生成N个点的平面坐标，应用分治法编程计算出所有点对的最短距离。

4. 分别对N=100000—1000000，统计算法运行时间，比较理论效率与实测效率的差异，同时对蛮力法和分治法的算法效率进行分析和比较。

5. 如果能将算法执行过程利用图形界面输出，可获加分。

**三、算法思想提示**

1.  预处理：根据输入点集S中的x轴和y轴坐标进行排序，得到X和Y,很显然此时X和Y中的点就是S中的点。

2.  点数较少时的情形

3.  点数|S|>3时，将平面点集S分割成为大小大致相等的两个子集SL和SR，选取一个垂直线L作为分割直线，如何以最快的方法尽可能均匀平分？注意这个操作如果达到效率，将导致整个算法效率达到θ。

4.  两个递归调用，分别求出SL和SR中的最短距离为dl和dr。

5.  取d=min(dl, dr)，在直线L两边分别扩展d，得到边界区域Y，Y’是区域Y中的点按照y坐标值排序后得到的点集（为什么要排序？），Y'又可分为左右两个集合Y’L和Y’R

6.  对于Y’L中的每一点，检查Y’R中的点与它的距离，更新所获得的最近距离，注意这个步骤的算法效率，请务必做到线性效率，并在实验报告中详细解释为什么能做到线性效率？

**四、实验要求**

1. 在blackboard提交电子版实验报告，注意实验报告的书写，整体排版。

2. 实验报告的实验步骤部分需详细给出算法思想与实现代码之间的关系解释，不可直接粘贴代码（直接粘贴代码者视为该部分内容缺失）。

3. 实验报告样式可从http://192.168.2.3/guide.aspx 表格下载－学生适用－在校生管理－实践教学－实验：深圳大学学生实验报告）

4. 源代码作为实验报告附件上传。

5. 在实验课需要现场运行验证。

1. **蛮力法求解最近点对**
2. 算法思想：遍历所有点对，计算所有点对距离，求出最小距。
3. 伪代码：

bruteforce(points, num)

*//points:点集, num:点数*

for i=1 to num

for j =i+1 to num

mindis = min(mindis, distance(points[i], points[j])

1. 算法复杂度分析：

遍历所有点对的操作次数为n\*(n-1)/ 2次，时间复杂度为O(n^2)

1. 数据测试与分析

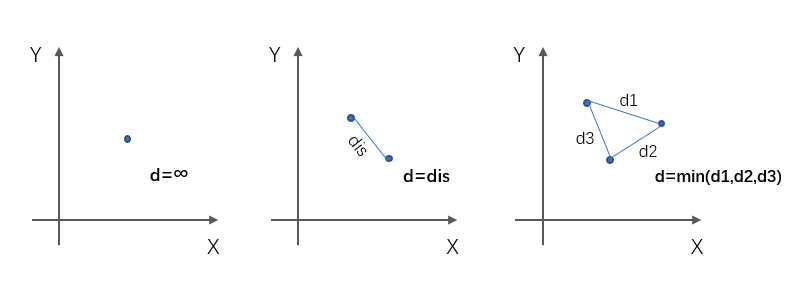
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 蛮力法求解最近点对问题 | | | | | |
| 数据规模 | 10000 | 20000 | 30000 | 40000 | 50000 |
| 实际时间/ms | 475.506 | 1897.26 | 4260.06 | 7592.63 | 11909.5 |
| 理论时间/ms | 476.38 | 1905.52 | 4287.42 | 7622.08 | 11909.5 |
| 数据规模 | 60000 | 70000 | 80000 | 90000 | 100000 |
| 实际时间/ms | 17157.8 | 23369.9 | 30538.9 | 38634.6 | 47717.4 |
| 理论时间/ms | 17149.68 | 23342.62 | 30488.32 | 38586.78 | 47638 |

表1 蛮力法数据测试和处理

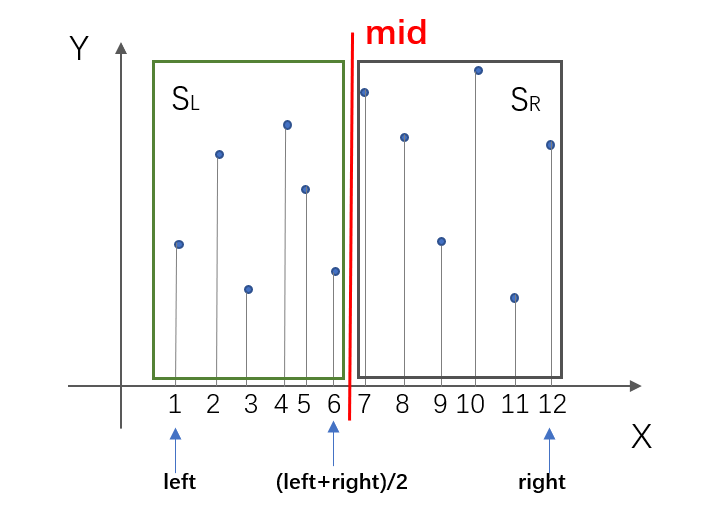
图1 蛮力法实际运行时间与理论时间拟合图

**分析：**理论时间的计算以表1数据规模50000的实际运行时间作为基准点，得到图1的实际时间曲线和理论时间曲线，实际时间和理论时间基本一致，两曲线拟合效果很好，基本符合O(n^2)的算法复杂度。

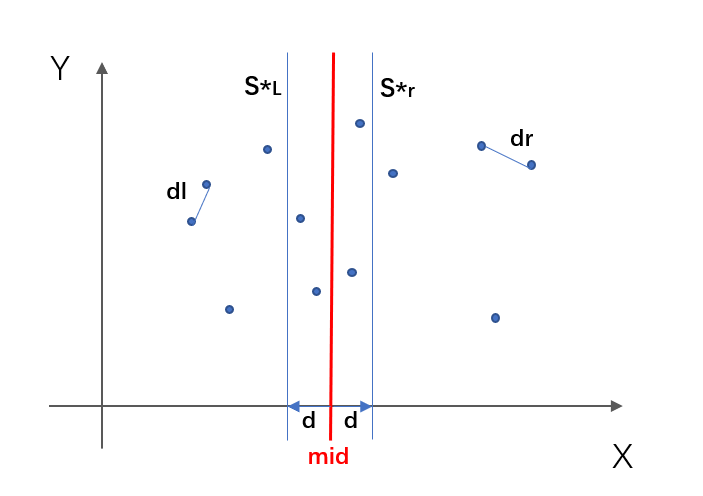
1. **分治法求解最近点对**
2. 算法思想：
3. 预处理：采用线性表保存所有点，对点集S按x轴升序进行快排，x轴有序便于快速均分点集。
4. 点数较少时：当子集S点数不大于3时，直接枚举得到最小距



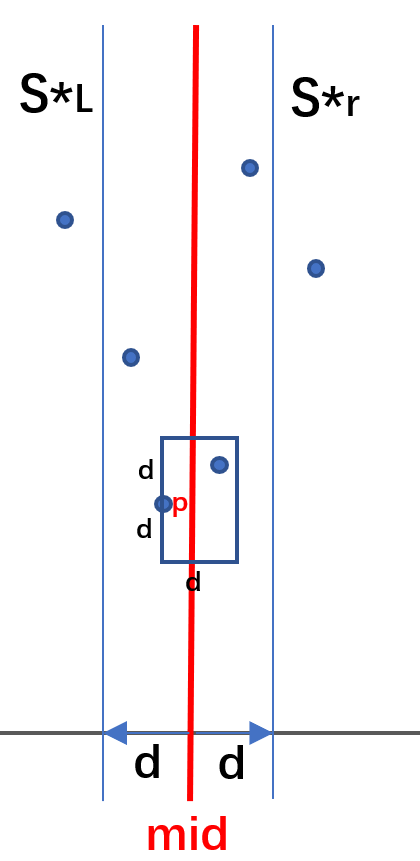
1. 当子集S点数大于3时，由于预处理后的点集在x轴有序，直接按表示当前点集的线性表中点分割成SL，SR，可实现快速均分点集。



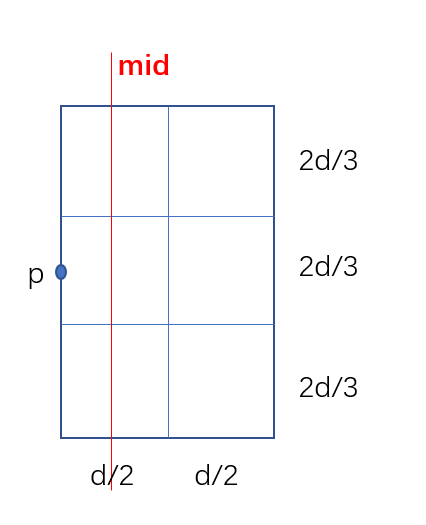
1. 递归调用，分别求解SL的最小点距dl，SR的最小点距dr
2. “治”部分，取d = min(dl, dr)，以mid为轴，向左右分别扩展d，得到S\*L和S\*R，求解[mid-d, mid+d] 区间内的最小点距



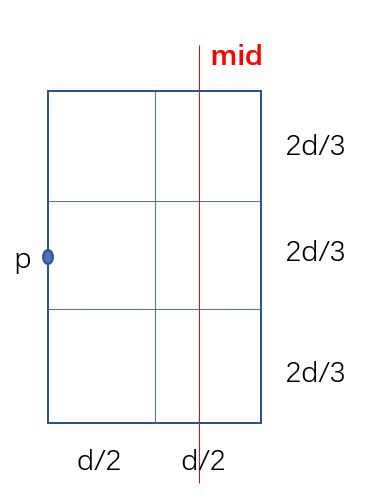
1. 遍历S\*L中的点，检查S\*R中的点距当前遍历点p的距离，更新最小距。由于已知当前最小距为d，所以只需检查S\*R中位于 以p为左侧中点的2d\*d矩形内的点。



1. 步骤6线性效率证明：
2. 将以p为左侧边中点的2d\*d矩形六等分为d/3\*2d/3 的小矩形，已知S\*L和S\*R中最小距为d，S\*R Y轴有序，得出结论，每次最多遍历 S\*R六个点
3. 假设：假设需遍历S\*R七个点
4. 情况1：mid切割线在矩形中线左侧，那么矩形右部分存在某个小矩形内右两个点，因为小矩形内最大距离为 5d/6<d ，与S\*R中最小距离为d的前提矛盾



1. 情况2：mid切割线在矩形中线右侧，则每个小矩形内含有两个点，显然不成立，易得此时mid右侧部分矩形最多有三个点



1. 假设推翻，原先结论正确，每次最多遍历S\*R六个点，内层循环为常数，于是为线性效率。
2. 算法实现：

分治算法最重要的是在合并部分尽可能提升效率，从而提升整个算法的效率。在更新最小距时已做到线性，此时最重要的是如何在合并时如何得到Y轴有序，在此我采用两种不同的解决方案。

1. **采用归并排序对区间的点进行Y轴排序（Method1）：**

由于归并排序本身就是分治法，通过备份点集，在合并求解最短距的同时，对备份点集按y轴排序，可以实现在合并区间内点集的y轴有序，使整个合并求解过程达到O(n)的时间复杂度

1. 伪代码：

merge(points,left,mid,right,d)

*//num:the number of points*

*//tmppoints:copy of points*

merge tmppoints[left:mid] and tmppoints[mid+1:right]

if num<=3

return min\_distance

else

for i=left to right

if |tmppoints[i].x-points[mid],x|<=d

points\_in[top++]=tmppoints[i]

for i=1 to top

for j=i+1 to top

if points\_in[j]-points\_in[i]>d

break

d = min(d, distance(points\_in[i],points\_in[j]))

return d

divide(points,left,right) *//分:降低问题规模*

if left<right

mid = (left+right)/2

dl = divide(points,left,mid)

dr = divide(points,mid+1,right)

d = min(dl,dr)

return merge(points,left,right,d)

else

return max

1. 在n=20000下验证算法准确性：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 穷举法 | 8.26 | 8.17 | 9.41 | 12.20 | 1.84 |
| 分治法1 | 826 | 8.17 | 9.41 | 12.20 | 1.84 |
| 穷举法 | 3.02 | 2.02 | 0.78 | 1.92 | 1.55 |
| 分治法1 | 3.02 | 2.02 | 0.78 | 1.92 | 1.55 |

1. 算法复杂度分析：

递归深度为logn，合并效率为n，备份点集效率为n，整体效率为O(nlogn)

1. 数据测试与分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 分治法(Method1)求解最近点对问题 | | | | | |
| 数据规模 | 10000 | 20000 | 30000 | 40000 | 50000 |
| 实际时间/ms | 4.748 | 9.708 | 14.872 | 20.252 | 26.225 |
| 理论时间/ms | 4.464 | 9.601 | 14.992 | 20.547 | 26.225 |
| 数据规模 | 60000 | 70000 | 80000 | 90000 | 100000 |
| 实际时间/ms | 30.908 | 36.611 | 41.714 | 47.506 | 52.975 |
| 理论时间/ms | 32.000 | 37.856 | 43.782 | 49.769 | 55.810 |

表2 分治法(Method1)数据测试和处理（数据规模1w-10w）

图2 分治法(Method1)实际运行时间与理论时间拟合图（数据规模1w-10w）

**分析：**理论时间以数据规模50000的实际运行时间计算而得，如图二，理论值和实际值曲线能较好的拟合，基本符合线性增长，但是理论值略大于实际值。可能对于该算法效率而言，基准数据规模过小，受计算机性能影响较大，导致在该规模下运行时间偏大，所得理论值曲线有一定的偏差。于是将数据规模增大10倍，再做测试。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 分治法(Method1)求解最近点对问题 | | | | | |
| 数据规模 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 实际时间/ms | 53.816 | 110.75 | 171.204 | 231.527 | 294.71 |
| 理论时间/ms | 51.712 | 109.652 | 169.942 | 231.758 | 294.71 |
| 数据规模 | 600000 | 700000 | 800000 | 900000 | 1000000 |
| 实际时间/ms | 356.525 | 417.849 | 482.068 | 550.949 | 613.365 |
| 理论时间/ms | 358.565 | 423.173 | 488.425 | 554.239 | 620.554 |

表3 分治法(Method1)数据测试和处理（数据规模10w-100w）

图3 分治法(Method1)实际运行时间与理论时间拟合图（数据规模10w-100w）

**分析：**将数据规模扩大十倍后，理论值曲线和实际值曲线能够很好的拟合，算法效率 符合O(nlogn)增长，实验结果很好。

1. 在合并部分，筛出落在(min-d,min+d)的点，对落在(min,mid+d)的部分进行快速排序（Method2）
2. 伪代码：

merge(points,left,mid,right,d) *//治:合并求解最小距*

*//筛选出x轴在(mid-d,mid)内的点*

for i=mid to left

if points[mid]-points[i]<=d

points\_in[top++]=points[i]

bound=top *//记录边界*

*//筛选出x轴在(mid,mid+d)内的点*

for i=mid+1 to right

if points[i]-points[mid]<=d

points\_in[top++]=points[j]

quicksort(points\_in,bound,top)

for leftpoint in points\_in(0,bound)

frame //以leftpoint为左侧边中点的2d\*d边界

for rightpoint in points\_in(bound,top)

*//更新最小点对距*

if rightpoint in frame

d = min(d,distance(leftpoint,rightpoint)

return d

divide(points,left,right)

if num<=3

return min\_distance

else

dl = divide(points,left,mid)

dr = divide(points,mid+1,right)

d = min(dl,dr)

return merge(points,left,mid,right,d)

1. 在n=20000下验证算法准确性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 穷举法 | 8.86 | 4.79 | 9.37 | 6.65 | 4.34 |
| 分治法2 | 8.86 | 4.79 | 9.37 | 6.65 | 4.34 |
| 穷举法 | 1.64 | 7.93 | 6.03 | 5.98 | 7.97 |
| 分治法2 | 1.64 | 7.93 | 6.03 | 5.98 | 7.97 |

1. 算法复杂度分析：

递归深度为logn，在合并部分，S\*R部分进行快速排序，快速排序的平均代价为nlogn，所以整个算法时间复杂度为O(n(logn)^2)

1. 数据测试和分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 分治法(Method2)求解最近点对问题 | | | | | |
| 数据规模 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 实际时间/ms | 65.9116 | 129.614 | 208.135 | 275.688 | 366.462 |
| 理论时间/ms | 64.3031986 | 136.3492738 | 211.3178495 | 288.1843009 | 366.462 |
| 数据规模 | 600000 | 700000 | 800000 | 900000 | 1000000 |
| 实际时间/ms | 443.636 | 511.819 | 582.158 | 683.508 | 782.204 |
| 理论时间/ms | 445.8643289 | 526.2019001 | 607.3401085 | 689.1783095 | 771.6383832 |

表4 分治法(Method2)数据测试和处理（数据规模10w-100w）

图4 分治法(Method2)实际运行时间与理论时间拟合图（数据规模1w-10w）

**分析：** 理论时间以数据规模为50000下的实际时间按n(logn)^2增长规律计算而得,拟合效果较好，但是实际值比理论值偏小。因为在合并部分进行的快速排序规模并不是全部，而是区间内的一小部分数据，所以该算法效率的上界是n(logn)^2，但是会略低于该理论效率。

1. **穷举法，两种分治算法效率比较**
2. 三种算法在数据规模为1w-10w的效率比较

**分析：**由上图可以非常明显观察到穷举法的运行时间远高于两种分治法，且随着数据规模的增长，与其他两种分治法的运行时间差距越来越大。

1. 两种不同理论效率的分治法在数据规模1w-10w下的比较

**分析：** 从图表上看，很明显在合并部分做到线性的Method1效率优于Method2(合并部分效率为nlogn)，符合预期猜想。

1. 两种不同理论效率的分治法在数据规模10w-100w 的比较

**分析：**将数据规模扩大十倍，Method2更偏向多项式增长，运行时间随数据规模扩大增长速度变快。反观Method1，由前面的线性拟合分析可知，随着数据规模扩大，Method1符合线性增长规律，所以较于上图，可以观察到两曲线差随着数据规模的扩大，曲线距差越来越大。

1. **算法选择及实验总结**
2. 由（三）的比较可得，穷举法思路很简单，但是效率非常低，适用于极小数据规模，在解决实际问题中价值不大。
3. 两种分治算法的抉择：
4. 从时间复杂度考虑，分治算法1(理论nlogn)更加具备优势。
5. 从空间复杂度考虑，分治算法1(理论nlogn)由于采用了归并排序对y轴进行排序同时保持点集y轴有需，所以需要备份点集，同时归并排序需借助辅助空间，所以空间复杂度较高。所以在可接受的时间内，在限定的内存环境下可以优先选择分治算法2
6. 本次实验发现并更正在实验1的一个错误，实验1中使用clock计时，而clock只精确到ms，而在统计算法效率时以ms为单位，从而导致误差累加。在本次实验中使用system\_clock精确计时。
7. 对于随机数的生成：由于本次实验需要用到随机浮点，但是rand()生成的浮点数质量并不好，从而使用C++11的随机数引擎default\_random\_engine生成分散度高的点集
8. 分治算法的合并部分是体现出整个分治算法效率的关键，在本次实验中采用两种合并测量(合并效率分别为n和nlogn)，通过对算法性能的比较，观察到合并部分效率的差距随着数据规模的扩大会进一步放大，所以在利用分治算法解决问题时需要尽可能采用最优的合并策略。