

# 数据结构大作业

# Data Structure Project

时空数据管理

Spatio-temporal Data Management

李英伟  
14307130092

December 19, 2015

## Contents

<b>1 [显著] 独立性声明</b>	<b>4</b>
<b>2 地图数据的抓取和处理</b>	<b>4</b>
2.1 地图数据结构的建立和地图的绘制 . . . . .	4
2.1.1 做法概要 . . . . .	4
2.1.2 亮点 . . . . .	4
2.1.3 缺点和改进方法 . . . . .	5
2.2 最短路算法的实验 . . . . .	6
2.2.1 做法概要 . . . . .	6
2.2.2 性能对比分析 . . . . .	6
2.2.3 亮点 . . . . .	7
2.2.4 待改进点 . . . . .	7
2.2.5 效果展示 . . . . .	8
<b>3 出租车轨迹数据管理</b>	<b>9</b>
3.1 做法概要 . . . . .	10
<b>4 范围查询</b>	<b>11</b>
4.1 实验方法概述 . . . . .	11
4.2 R 树 . . . . .	11
4.3 线段树套红黑树 . . . . .	11
4.4 线段树套有序数组 . . . . .	12
4.5 fractional cascading . . . . .	12
4.6 效率对比 . . . . .	12
4.7 范围查询在地图中的应用 . . . . .	13
<b>5 近邻查询</b>	<b>14</b>
<b>6 挖掘和分析——哪里更能拉到客?</b>	<b>15</b>
6.1 功能描述 . . . . .	15
6.2 实现方法——三维 kd 树 . . . . .	16
6.3 效果展示 . . . . .	16
<b>7 其它</b>	<b>17</b>
<b>8 [附录一] 实验环境以及相关配置说明</b>	<b>17</b>
<b>9 [附录二] 软件使用说明</b>	<b>18</b>

<b>10 [附录三] 文件总览</b>	<b>20</b>
<b>11 [附录四]fractional cascading 优化</b>	<b>20</b>

## [显著] 独立性声明

除了以下提到的以外都是自己独立完成的：

1. pugixml 是一个解析 XML 的库，官方网站：[pugixml.org](http://pugixml.org)
2. 使用了 boost 库中的 rtree
3. plot\_config.xml、ShortestPath.conf 两个文件分别是用来配置地图配色和车速的，修改自 [github.com/Orthocenter/YuxinMap](https://github.com/Orthocenter/YuxinMap)
4. 实验中所使用的红黑树调用了 STL 中的 map
5. 使用了 opencv2 库来画点和直线
6. 在自己实现三维 kd 树前曾经问过林禹全同学二维 kd 树的做法，并且看了他二维 kd 树的实现，所以代码可能会有点相似，但是也是自己独立完成的，毕竟二维 kd 树复制过来是不能变成三维的。在这里非常感谢林禹全同学的帮助。

## 地图数据的抓取和处理

### 地图数据结构的建立和地图的绘制

#### 做法概要

在读取上海地图时使用了 pugixml 对上海地图文件进行了解析。使用了 boost 库中的 R 树和 std 库中的 map 对读取出的点和道路的数据进行预处理并且存储。当请求一块范围内的地图时（即给出矩形对角线上两点的经纬度坐标时），通过之前的数据结构得到要画线和多边形的信息，通过 opencv2 库中的画点和画多边形的函数，最终画出地图。

#### 亮点

1. 通过 R 树可以快速的查出要画的小区域中的点，并且通过预处理 O(1) 的找到与相邻的点构成的一小段路的相关信息，即可

快速画出对应小区域中的信息，而不用把全图都扫一遍。

2. 地图层次分明。地图是有层次的，如高架路在最上面，下面是普通的路，再下面是地下通道，最后是河流等。为了画出层次正确的地图，把找到的线和多边形按预处理出来的层次进行排序，按层次从低到高依次绘制。为了描出路的边，可以先画一条比路粗两个单位的线，再在上面画路，就可以达到描边的效果，放在有层次的地图中要注意对于相同层次，要统一先画底色，再统一画线，这样可以表现出同一层次道路相互交叉的效果。
3. 配色美观。效果见 Figure1。

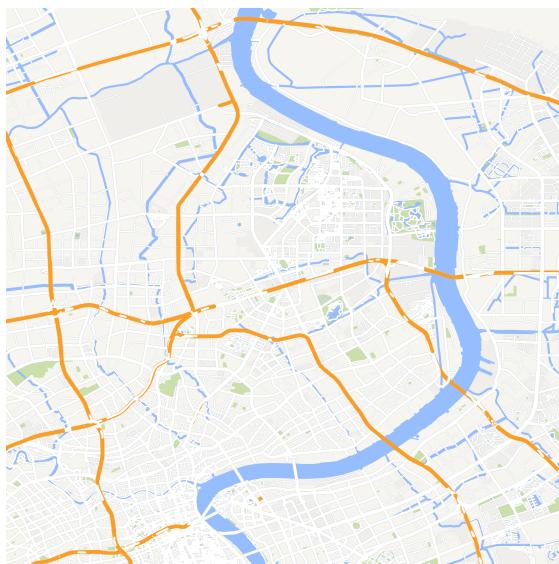


Figure 1: 地图效果图

### 缺点和改进方法

因为时间有限，不可能做到尽善尽美，所以有些有待解决的次要一些的问题只能提供一个解决方法。

1. 索引选择的数据结构有待改进：R 树虽然很强大，可以支持动

态插入点、插入矩形框。在实际应用环境中如果地图是每天动态更新的，那么 R 树是会有优势的。但是如果对地图的更新度的要求不是很高，我们完全可以采用静态的数据结构，可以达到更快更好的效果，在报告的后半部分详细的讨论了高维空间索引的问题。以后有时间可以考虑换一种更合适的数据结构。

2. 地图的渲染速度较慢，原因有三：一是绘图比较细致，为了描出道路的边几乎花了二倍的时间；二是 CPU 渲染速度较慢，可以考虑使用 GPU 绘图；三是当加载全图时把所有细节都画了出来，为了加速可以只画主要的路和建筑物。

## 最短路算法的实验

### 做法概要

本 PJ 实现了三种不同的最短路算法（SPFA、dijkstra、A\*），得到了两种意义上的最短路：路程最短的路和所花时间最短的路。两者的不同之处在于前者的边权是两点之间的实际距离，而后的边权是两点之间的实际距离除以依据这条道路等级而假设的速度（即经过这段所需要的时间）。在所花时间意义下得最短路还有躲避拥堵的功能，即如果指定某条路为拥堵路段，那么这条路上行驶的速度就会减小到一个很小的值。有一个想法是把图中入度为 2 的点去掉，这样道路上的总点数就会从 25000 个降到 8000 个，这样虽然可以降低计算最短距离的时间，但是把点画在图上时会比较复杂，而且因为点数过少，所以不能很好的比较各算法之间的时间复杂度。

大致了解了孙老师上课时推荐的 SIGMOD2013 的最短路算法 [1]，通过和张睿哲同学的讨论认识到这种算法由于占用空间过大，并不能在普通的笔记本电脑上进行实验。

### 性能对比分析

SPFA、Dijkstra 和 A\* 都是最平常的做法，不再赘述，其中 A\* 所使用的估价函数是两点之间直线距离（路程最短路）和两点之间最短距离除以最高车速（时间最短路）。

性能实验方法：随机在路上找起点和终点，然后分别用六种方法（时间意义下和路程意义下得 SPFA、Dijkstra 和 A\*）进行最短路的

计算，循环一定次数后得到平均运行时间，结果如下：

循环次数	100 次	500 次	1000 次
[SPFADist]	4213ms	3796ms	4593ms
[DijkDist]	72ms	66ms	85ms
[A* Dist]	19ms	18ms	23ms
[SPFATime]	1982ms	2176ms	2201ms
[DijkTime]	59ms	65ms	71ms
[A* Time]	25ms	24ms	32ms

可以看出总体而言 SPFA 表现最差，A\* 表现最好。在距离意义下 A\* 是 Dijkstra 的 4 倍，而在时间意义下 A\* 是 Dijkstra 的两倍多，原因是上述估价函数是基于距离进行估价的，所以在时间意义上表现不够好。也曾经尝试过改变 A\* 的估价函数，试图加上速度这一因素，但是效果还没有最初的效果好。

## 亮点

1. 两种最短路和避免拥堵的情况都考虑到了实际情况的应用，路程最短路可以作为步行的结果，时间最短路可以作为交通工具为汽车时的结果，还考虑了堵车的情况。
2. 考虑了单行道，能否掉头，能否左转等情况，不会出现违反交通规则的路线。

## 待改进点

1. 最短路算法中堆的实现采用了 stl 中的 `priority_queue`，虽然复杂度上没有变化，但是因为没有及时删除堆内无用元素，所以常数比较大，并没有达到最优的性能。以后可以考虑手写堆，或者用斐波那契堆。
2. 没有尝试双向 Dijkstra 等算法。



Figure 2: 邯郸到张江最短路

### 效果展示

从张江校区往返邯郸校区，考虑当前杨浦大桥是否拥堵的四种结果如下图所示。可以看出步行距离大概是 18km，假设全部道路顺畅的话大概需要 23 分钟，如果杨浦大桥拥堵，那么到张江需要 29 分钟，从张江出发需要 27 分钟。而且四幅图走的路线也不完全一样。上面的时间(可能会稍短一些，因为考虑的是道路非常通畅时的情况)和距离和高德地图给出的基本一致。



Figure 3: 邯郸到张江最短路（躲避拥堵的杨浦大桥）



Figure 4: 张江到邯郸最短路



Figure 5: 张江到邯郸最短路（躲避拥堵的杨浦大桥）

## 出租车轨迹数据管理

## 做法概要

根据出租车轨迹文件的特点，直接用数组存储，二分查询即可<sup>1</sup>。经统计可知一共有 1707 辆出租车的 13294680 条左右的采样点信息。所以如果只是查询一条轨迹并且把它显示出来的话并不需要特殊的数据结构，只要二分出某一出租车数据开始的位置，然后取出需要的兴趣点即可。实测这样查询一条完整的出租车 24 小时的路径只需要 0.7ms 的时间。最终实现的功能是可以查询某一出租车在一段时间之内的轨迹数据并且显示在图上，还可以查询第 i 个数据点的相关信息。显示效果如图。

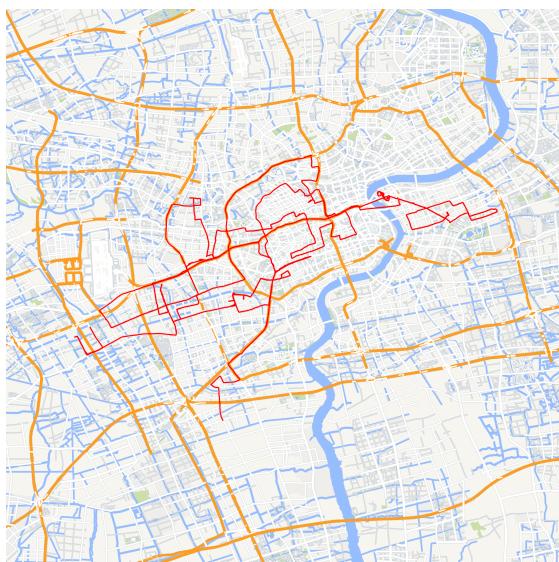


Figure 6: 10944 号出租车下午 1 点到晚上 11 点的行驶轨迹

虽然本题可以这么简单的实现，但是这和实际情况并不完全符合，因为分析数据时并不会只查询一条出租车的路径，而会像之前我处理地图数据时一样查询给定二维空间内的数据点，甚至是三维空间（加上时间这一维，在最后一题中会有涉及）内的数据点。再加上后面的任務中也涉及到范围查询的问题，所以我认为仅仅调用 R 树的库是不足以说明问题的，所以我写了几种数据结构来对比它

<sup>1</sup>当然使用 1707 个 vector 存储就不用二分，直接按地址查找就好了，但是这样也会涉及 ID 和下标之间的映射问题，并不一定会快很多

们的性能，详细分析见范围查询一节。

## 范围查询

### 实验方法概述

因为地图中只有 50000 个点，为了在不同规模的数据集上进行测试，我选择了给出的出租车数据文件。出租车数据文件大小为 1G，有大约 13293000 个点，可以通过截取数据的不同部分来获得不同规模的数据。编译时开 O2 优化。

## R 树

调用了 Boost 库中的 R 树，R 树的结点分裂算法有线性、二次方和 R\* 三种，叶节点个数 M 也是可以修改的。下面用 500 万个点随机询问 10000 次的时间来代表查询性能。

结点分裂算法	linear	quadratic	rstar
建树时间	6s	7s	17s
查询时间	28s	26s	44s

上面实验是在叶节点个数为 16 时得到的，通过对叶节点个数的实验发现，叶节点个数大于 20 时查询时间都差不多逐渐趋近于 24s，但是建树时间越来越长。最后选择了 20 个子节点的 quadratic。

## 线段树套红黑树

按照坐标点的横坐标为关键字建立一棵线段树，在每个线段树区间的节点上挂一棵以纵坐标为关键字的红黑树（红黑树中的元素满足横坐标在线段树区间内）。查询时通过线段树可以找到满足在横坐标范围内的点，再通过红黑树找到纵坐标在查询范围内的点。

这样的话空间复杂度为  $O(n \log_2 n)$ ，因为每个节点都最多出现在  $\log_2 n$  个区间中，所以最多出现在  $\log_2 n$  棵红黑树中。

查询的时间复杂度为  $O(\log_2 n * \log_2 n)$ ，因为最多进入  $\log_2 n$

个结点在红黑树中查找需要  $O(\log_2 m) \leq O(\log_2 n)$  ( $m$  为红黑树结点个数) 所以上面提到的复杂度其实是一个上界，实际复杂度可能会比这个小。

## 线段树套有序数组

将其实现后稍加思考可以发现，上面做法中的红黑树的用处只是找到纵坐标满足条件的点，所以如果用有序数组替换红黑树只要用二分查找的方法就可以达到同样的功能，虽然复杂度不变，但是常数应该会比红黑树小很多。

## fractional cascading

fractional cascading[2] 是一种基于上述数据结构的优化技巧，通过一定的预处理，可以把时间复杂度优化到  $O(\log_2 n)$ 。而且这种优化可以很容易的扩展到多维的情况，即  $k$  维的复杂度为  $O(\log_2 n^{k-1})$ 。

主要想法是发现其实在线段树结点内的每次二分查找并不是必要的，因为每次二分查找所要找的纵坐标是一样的，所以可以通过线段树结点的父结点查找出的边界直接  $O(1)$  的给出子结点的边界。

具体详细做法参见附录。

## 效率对比

序号	算法	点数	询问数	询问时间
1	线段树套红黑树	13293000	1000	63s
2	线段树套有序数组暴力	13293000	1000	25s
3	线段树套有序数组二分	13293000	1000	16s
4	fractional cascading	13293000	1000	15s
5	R 树	13293000	1000	53s

序号	算法	点数	询问数	询问时间
1	线段树套红黑树	5000000	1000	19.7s
2	线段树套有序数组暴力	5000000	1000	6.9s
3	线段树套有序数组二分	5000000	1000	5.3s
4	fractional cascading	5000000	1000	4.9s
5	R 树	5000000	1000	18.5s

实验的结果和我想象的基本相同，但是 fractional cascading 的效果不如我想象的好。

我觉得可能的原因有这么几点：一是程序 3 的空间复杂度虽然并不增加，但是常数大概是别的程序的 3 倍，因为点数很多所以会更多的使用虚拟内存（本机内存共 8GB，当点数为 5000000 时已经占用了 5GB 的内存，当点数为 13293000 时占用了 12GB 内存），因为磁盘的读取速率比内存慢很多，所以程序运行速度会因此而变慢，可以推测出如果在内存够用的情况下程序 4 是会优于程序 3 很多的，但是如果使用较小数据来测试发现两者（程序 3, 4）所花的时间几乎是一样的，没有太多的差异；二是程序的测试数据是随机生成的，所以数据中询问的范围会比较大，这样会在时间复杂度中添加  $O(\text{查询所得点数})$ ，使得最后得到的询问时间加了这一项后显得不正常。

通过以上分析，基本可以认为 fractional cascading 的效果虽然没有想象的好（像分析的一样去掉一个  $\log$  的效果），但是也比实验上看上去的要好。

通过这个实验我明白了：程序的运行时间确实受到很多条件的影响，虽然最后结果和预想的不完全一样，但是还是可以根据上面的分析说明加了 rational cascading 优化之后确实效果要比单独的二分查找要好一些。

总之，程序 4 最快，大概是 R 树的几倍，而且随着点数的增多优势在不断增大。红黑树的常数太大导致实际效果比 R 树还差。

## 范围查询在地图中的应用

如果地图上的兴趣点是动态增加的，而且增加的频率很快，且要求实时更新，那么应该选用 R 树这样的可以动态增删结点的数据结构。而如果是静态的或者是不要求实时性的就可以选用上面提到

的比较好写也比较快的数据结构。

在实际实现上还是选择了 R 树，是因为在分析各种范围查询复杂度之前已经使用了 R 树，就不想再换了。而且对于这个数据规模，各种数据结构之间的差异其实不是很明显，使用 R 树还可以支持随时插入兴趣点。但是如果有海量兴趣点的话就要考虑数据结构的选择问题了。因为地图数据中真正的兴趣点不是很多，所以把所有路上的结点作为兴趣点来使用。范围查询效果图如图 5 所示。

因为选择了 R 树，所以同时实现了插入新兴趣点的功能。

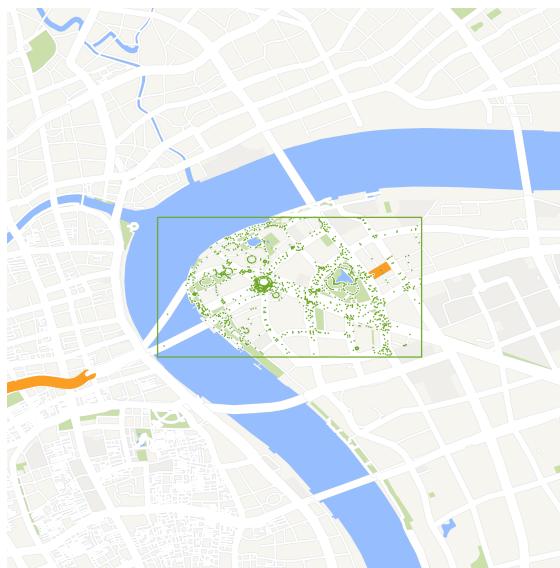


Figure 7: 范围查询效果图

## 近邻查询

近邻查询使用的也是 R 树实现的，支持 K 近邻查询，查询所需时间和 k 有关，具体关系如下：

k	1	10	100	1000	10000	100000	1000000
time(ms)	0.026	0.036	0.1	0.578	5.80	20.25	16.13

在地图上显示效果如图所示。红圈为查询的中心点，绿色的点为周围的 1000 个近邻点。

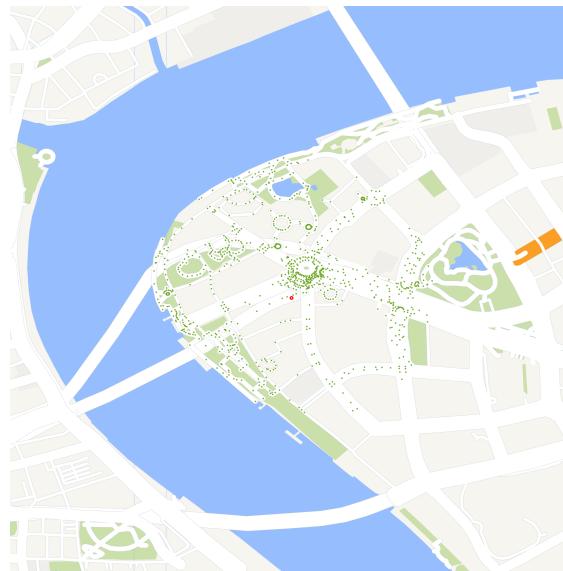


Figure 8: k-邻近查询效果图

## 挖掘和分析——哪里更能拉到客？

### 功能描述

针对出租车司机经常有载不到客的情况，我设计的一套系统可以基于出租车的轨迹数据为在某一特定时间特定地点的空车司机推荐附近更容易拉到客人的点。具体来说就是给出当前点和当前时间就可以给出附近更容易接到客人的点，并且显示在地图上。

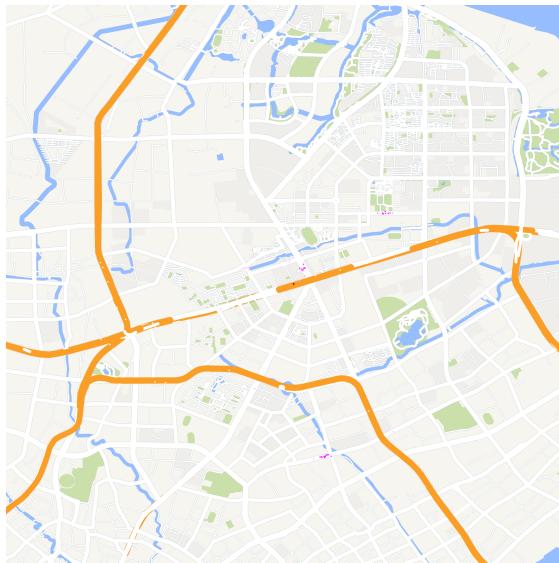


Figure 9: 拉客位置推荐-中午一点复旦大学周边

### 实现方法——三维 kd 树

完成这一任务主要使用的数据结构是三维 kd 树，三维是时间的一维和空间的二维。首先把所有载客点加入 kd 树中，载客点的判断标准是这个时刻的点不是空车，而上个时刻是空车。当获取到一次询问时查询在这个时间点周围一段时间内的载客点。然后对每个载客点算出在它周围邻域内载客点的密度，最后选密度大的一些点作为推荐。更具体一点是如果周围点数高于平均，那么用周围点个数减去平均值的二倍再除以最大值乘以六；否则取负无穷。然后放到 sigmoid 函数里面，目的只是使大于某一阈值的点尽可能红，其它点有从红到白的渐变而已。

### 效果展示

图为在中午 13 点在复旦大学的询问请求，点的桃红色颜色越深就代表自信度越高。通过看图可以发现周围三个主要的接客点分别是五角场万达广场门口、控江路江浦路的上海新华医院以及长海路

黑山路上的长海医院，而其他时段就不一样了。中午在五角场和医院打车的人多都非常好理解。

## 其它

通过后缀数组实现了兴趣点名字的查询，比如输入“小学”就会查到所有名字中带有“小学”的兴趣点坐标。

做的是命令行 ui，操作起来相对麻烦一些。

## [附录一] 实验环境以及相关配置说明



Figure 10: 笔记本电脑相关



Figure 11: IDE 环境

```
# taxi git:(master) ✘ g++ ~v
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr
--with-gxx-include-dir=/Applications/Xcode.app/Contents/Developer/Platforms
macosx.platform/Developer/usr/include/5.0.0/xcode-11.1.sdk/usr/include/c++/4.2.1
Apple LLVM version 7.0.2 (clang-700.1.81)
Target: x86_64-apple-darwin15.2.0
Thread model: posix
```

Figure 12: 编译器

相关环境如图所示，之后需要安装 opencv2 和 boost 库，最后直接把 LiMap、NearestNode.conf、plot\_config.xml、shanghai\_map.xml、shanghai\_taxi\_20150401.csv 放到一个文件夹下，最后从命令行中打开这个文件夹最后输入命令./LiMap 即可运行。

## [附录二] 软件使用说明

打开软件后输入 help 查看，内容如下：

显示地图：

语法： showmap lat lon level

解释： level 是缩放级别，取值为 11 到 18

举例：

showmap 31.30186 121.51159 13

查最短路：

语法 1： shortestpath id1 id2 n ...

解释 1： id1 id2 为两个 node 的 id，n 为拥堵路的数量，后面跟 n 个数代表拥堵路的 id

举例 1： shortestpath 1008057714 1813887498 2  
227962891 11635095

语法 2： shortestpath s1 s2 n ...

解释 2： s1 s2 是点的名称，其它同上

举例 2： shortestpath 邯郸路 张江校区 2  
8886548 8886576

通过名字找坐标和 id：

语法： queryname name

举例： queryname 复旦

查询道路：

语法： queryway name/id

举例： queryway 肇嘉浜路

queryway 世纪大道

queryway 272231909

范围兴趣点查询：

语法： querynode maxlat minlon minlat maxlon

举例： querynode 31.2449 121.4886 31.2349

121.5073

k邻近兴趣点查询：

语法：querynearest lat lon k

举例：querynearest 31.2392 121.4956 100

最短路性能测试：

语法：shortesttest n

说明：随机生成起点和终点，测试n次不同最短路  
算法

举例：shortesttest 20

插入兴趣点：

语法：insertpoint lat lon

举例：

querynode 31.2449 121.4886 31.24 121.49

insertpoint 31.2440 121.489

querynode 31.2449 121.4886 31.24 121.49

以下为出租车相关，在使用以下功能前先输入loadtaxi  
载入出租车轨迹信息

查询一辆出租车轨迹：

语法：querytaxi taxiid starttime endtime n

...

说明：n为要查询的点的个数，后面n个数字，代表  
查询第几个点

举例：querytaxi 10944 0:0:1 23:59:59 2 2 3(

最后三个数字代表，输出两个点的信息，分别是  
第二个点和第三个点)

查询去哪里载客：

语法：queryneartaxi lat lon r time duration

说明：r是搜索半径 time是当前时间 duration是  
设置搜索范围的时间长度

举例：queryneartaxi 31.2363 121.4690 0.01

21:0:0 3600

上例为当前时间为21点，所以搜索范围为20点半到  
21点半

输出得图片可以在同一文件夹中找到对应或相似的文件名

## [附录三] 文件总览

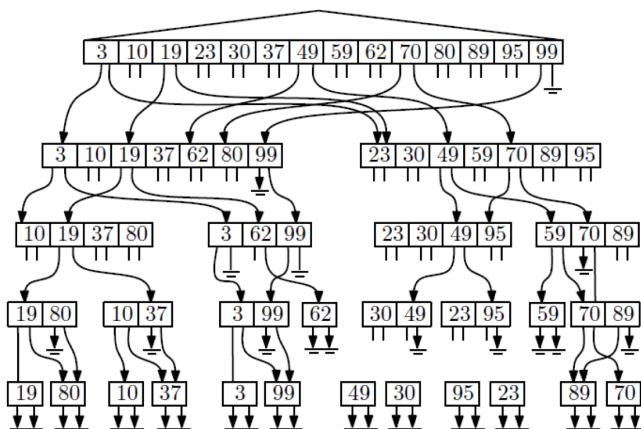
如图 13、图 14。

## [附录四] fractional cascading 优化

先考虑这么一个问题，假设有两个数组  $A_1$  和  $A_2$  且  $A \subseteq B$ ，我们要在  $A$  和  $B$  中查找大于等于  $x$  的元素有多少。朴素的方法是二分两次。但是其实通过预处理给  $A$  中每一个元素一个指向  $B$  中元素的指针满足如果  $a_i -> b_j$  那么  $a_i \leq b_j$  且  $a_i > b_{j-1}$  即  $a$  指向  $b$  中大于或等于  $a$  的第一个元素。

对于线段树的每个节点  $v$ ，套一个单调数组  $p$ ，记录的是  $y$  坐标，对于  $p$  中每个元素附带两个指针  $\text{ptl}$  和  $\text{ptr}$ ，分别指向其左儿子中不比  $p[i]$  小的第一个元素和其右儿子中不比  $p[i]$  小的第一个元素。

### Fractional cascading



名称	修改日期	大小	种类
▼ KDtree		--	文件夹
kdtree.cpp	昨天	3 KB	C++ Source File
kdtree.h	昨天	1 KB	C Header Source File
KDtree.pro	昨天	375 字节	Qt Project File
KDtree.pro.user	昨天	19 KB	文稿
main.cpp	昨天	118 字节	C++ Source File
▼ LiMap	下午8:14	--	文件夹
cmd.cpp	下午6:46	11 KB	C++ Source File
kdtree.cpp	上午11:03	3 KB	C++ Source File
kdtree.h	上午11:03	1 KB	C Header Source File
▼ Libs	15/11/25	--	文件夹
► pugixml	15/12/2	--	文件夹
LiMap.pro	上午11:03	1 KB	Qt Project File
LiMap.pro.user	下午8:14	19 KB	文稿
load.cpp	上午11:03	10 KB	C++ Source File
main.cpp	下午5:24	2 KB	C++ Source File
nearest.cpp	15/12/13	545 字节	C++ Source File
NearestNode.conf	15/11/25	3 KB	Configuration file
plot.cpp	上午11:03	14 KB	C++ Source File
ploymethods.cpp	上午11:03	5 KB	C++ Source File
query.cpp	15/12/13	220 字节	C++ Source File
sa.cpp	上午11:03	4 KB	C++ Source File
ShortestPath.conf	15/11/25	3 KB	Configuration file
shortestpath.cpp	上午11:03	13 KB	C++ Source File
taxi.cpp	上午11:03	3 KB	C++ Source File
visitprivat...lement.cpp	15/12/13	236 字节	C++ Source File
ywmap.cpp	15/12/12	645 字节	C++ Source File
ywmap.h	下午5:24	10 KB	C Header Source File
▼ Release	下午8:20	--	文件夹
► LiMap	下午8:17	465 KB	Unix executable
NearestNode.conf	15/11/25	3 KB	Configuration file
ShortestPath.conf	15/11/25	3 KB	Configuration file
▼ taxi	下午8:22	--	文件夹
dataSegmentor.cpp	上午11:03	278 字节	C++ Source File
main2.cpp	下午3:31	5 KB	C++ Source File
mainSegBF.cpp	上午11:03	5 KB	C++ Source File
mainSegSet.cpp	下午3:50	4 KB	C++ Source File
mainWithOut2.cpp	下午3:31	5 KB	C++ Source File
tastcasemaker.cpp	上午11:03	714 字节	C++ Source File

Figure 13: 文件总览

主要文件名称	主要功能	大小	行数
<b>cmd.cpp</b>	命令行ui	11KB	315
<b>kdtree.cpp</b>	kdtree实现	3KB	101
<b>kdtree.h</b>	KDtree头文件	1KB	51
<b>load.cpp</b>	载入地图文件并存入相关数据结构中	10KB	275
<b>main.cpp</b>	主文件	2KB	53
<b>nearest.cpp</b>	查k邻近点相关	0.5KB	17
<b>plot.cpp</b>	画图相关	14KB	441
<b>plotmethods.cpp</b>	画图基础函数封装	5KB	124
<b>query.cpp</b>	范围查询相关	0.2KB	8
<b>sa.cpp</b>	后缀数组相关	4KB	164
<b>shortestpath.cpp</b>	最短路相关	13KB	421
<b>taxi.cpp</b>	出租车相关	3KB	97
<b>visitprivateelement.cpp</b>	用来访问类中私有元素	0.2KB	13
<b>ywmap.cpp</b>	公共函数	0.6KB	26
<b>ywmap.h</b>	头文件	10KB	239
<b>dataSegmentor.cpp</b>	截断处理出租车文件	0.2KB	16
<b>main2.cpp</b>	加fractional cascading优化的线段树套有序数组	5KB	196
<b>mainSegBF.cpp</b>	线段树套有序数组暴力	5KB	209
<b>mainSegSet.cpp</b>	线段树套红黑树	4KB	171
<b>mainWithOut2.cpp</b>	线段树套有序数组二分	5KB	180
<b>testcasemaker.cpp</b>	测试数据制造器	0.7KB	31

Figure 14: 主要文件功能

这样，在查询时就不用每次都去二分了，而可以通过父节点的指针直接定位到第一个有效元素。这样只需要在根节点处二分就可以一劳永逸了。总的查询复杂度为  $O(\log_2 n)$ 。

## References

- [1] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. *CoRR*, abs/1304.4661, 2013.
- [2] Wikipedia. Fractional cascading — wikipedia, the free encyclopedia, 2014. [Online; accessed 19-December-2015].