

《数据结构》

课程设计报告

实验名称： \_\_\_出租车轨迹分析\_\_\_

专业班级： 计算机科学与技术1班

学号： 202166440366

姓名： 成云

得分：

日期： 2023.5.11

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

**一、需求分析**

根据《数据结构大作业任务书》（下称《任务书》）中“8.出租车轨迹分析”的数据集和要求，将任务分为偏向前端的“可视化”部分和偏向后端的“功能实现”部分，另外需要设计程序衔接二者。

“可视化”部分，利用高德地图的JavaScript API，编写HTML、CSS语句和JavaScript代码，重点实现接收用户需求，并将结果呈现在电子地图（网页）上。

“功能实现”部分，考虑C++实现。主要关注《任务书》中F3—F9部分，利用离散化、序列化等技术与标准库，依托哈希表、红黑树等数据结构，构建文件与内存的关系，在预处理数据后，建立函数逐个实现所需功能。

为了更好地衔接，在各自的程序中额外添加了关于文件读写、参数接收等内容。

另外，本设计中定义的北京范围按经纬度的角度，被划定为115.40-117.60经度范围以及39.40- 41.10的纬度范围所构成的一个矩形，矩形之外的地区被定义为北京外。

**二、详细设计**

**1. 可视化**

本次大作业初步打算是将F1、F2两个功能使用高德地图API接口来在浏览器网页端展示，后来为了将整个大作业系统化地使用，决定将之后的F3——F9全部使用HTML通过JavaScript语言在网页端实现包括了输入和输出的用户交互界面。

高德地图 JS API 是一套 JavaScript 语言开发的地图应用编程接口，移动端、PC端一体化设计，一套 API 兼容众多系统平台。在这里我们利用高德地图JS API，在JS端进行可视化。

在注册账号并申请Key后，可在本地和服务器应用高德地图API。

1. **F1：出租车轨迹可视化**

**[公共部分]**

使用此API功能，需要编写 HTML 页面。先创建初始地图图层，进行底图的初始化和图标样式设置。然后，对用户交互界面进行布局。需要一个输入卡片，设置两个按键，分别对应展示“全部车辆”和“某一台车辆”的轨迹点。并对其捆绑鼠标点击事件，链接到相关的功能函数showmark(int)。



图1.1.1 F1功能栏设置

功能函数showmark(int):通过鼠标点击事件所传递的参数，如果“显示全部车辆”，传参为0；“显示某一车辆”传参为用户所输入的数值。将参数传递到函数后，创建海量点对象，设置点样式。这里就需要使用到MassMarks海量点所要展现的数据集。

**[成云]**

遇到了问题：由于安全原因，浏览器不允许JavaScript在没有用户明确操作的情况下访问本地文件系统。所以读取本地文件就会有一定的困难。

解决方法：通过查找资料，寻找读取文件内容的方法，最终将10375个文本文件内容通过C++文件操作，先将原本的txt文本中除了坐标和车牌号之外的其他信息去除，然后将内容更改为以taxis为名的标记点数组形式：

var taxis =

[{"lnglat":[116.51172,39.92123],"name":"1","style":1},{"lnglat":[116.51135,39.93883],"name":"1","style":1},{"lnglat":[116.51135,39.93883],"name":"1","style":1},……]

然后，通过 const taxis = window.taxis;，使用window功能进行解析，将txt文件中的taxis数组传递到JS中的taxis变量，这样，就能把坐标点的信息读取到，进而在地图上进行点标记。

**[公共部分]**

读取后， 创建海量点对象，设置点样式。然后为了对显示的标记点所代表的车辆进行区分，我又设置了一个“鼠标监听事件”，当鼠标在地图上悬停时，若指针指到了某一标记点，则将该标记点所归属的车辆的信息所展示出来，如该车的车辆序号。



图1.1.2 海量点样式设置

**[柴豪]**

a. 显示所有车辆轨迹：在此步骤遇到了第二个问题，就是海量数据的规模太大。总共有超过一万个文本文件需要读取，这将会导致运行的速度非常之缓慢。所以，我尝试了方法一，先预处理，再将之前所更改了格式的一万个文本文件进行合并，直接归为一个文件，然后把所有车辆的标记点全都放到一个数组来表示。但是尝试之后发现这个方法并不可行，因为数据量太大，合并后得到的txt文件已经超过了JS环境所能读取的最大容量，所以运行时根本没有读取到数据。将所有的车辆的所有坐标汇总起来发现共超过了一千万的数量级。然而对高德API接口所提供的海量点标记函数，当需要在地图展示数量为十万以内的点并且需要较好的性能表现时，才能有一定的效果。所以最后只能使用方案二，逐一读取文件。这里就使用了一个回调函数方法来动态的加载JS脚本文件：function loadScript(src, callback)；然后通过字符串拼接的方法，利用循环，依次读取从taxi1一直到taxi10375.txt文件中的数据。每读一辆车的信息，就在地图上标记该车的所有标记点。

所以在此大作业中，“显示所有车辆的轨迹点”并未彻底实现，失败原因分析：弊端就是上面所分析到的，读取速度较慢，当数量级超过API所能承受的范围（十万）之后会导致网页崩溃。但是由于API本身提供服务限制的原因，此问题没有得到解决。另外，或许存在方法能够实现将千万级别的标记点在地图上展示，但是查阅资料的精力不足，导致并未查找到最终实现的方法。在程序中，“展示所有车辆轨迹”的函数功能有所阉割，只显示了部分车辆的轨迹点，数量级在十万以内。

**[成云]**

b.单独显示某一辆车的轨迹：此步骤较为简单，与显示所有车的轨迹方法相同，且数据量的数量级较小，速度快。程序只需要从用户输入的车辆ID获取到参数，然后进入到相对应的taxi-i.txt文件当中，进而读取数组内的坐标点信息，再通过window解析，使用海量点标记函数，就能得到最终展示效果。

展示效果图如下:

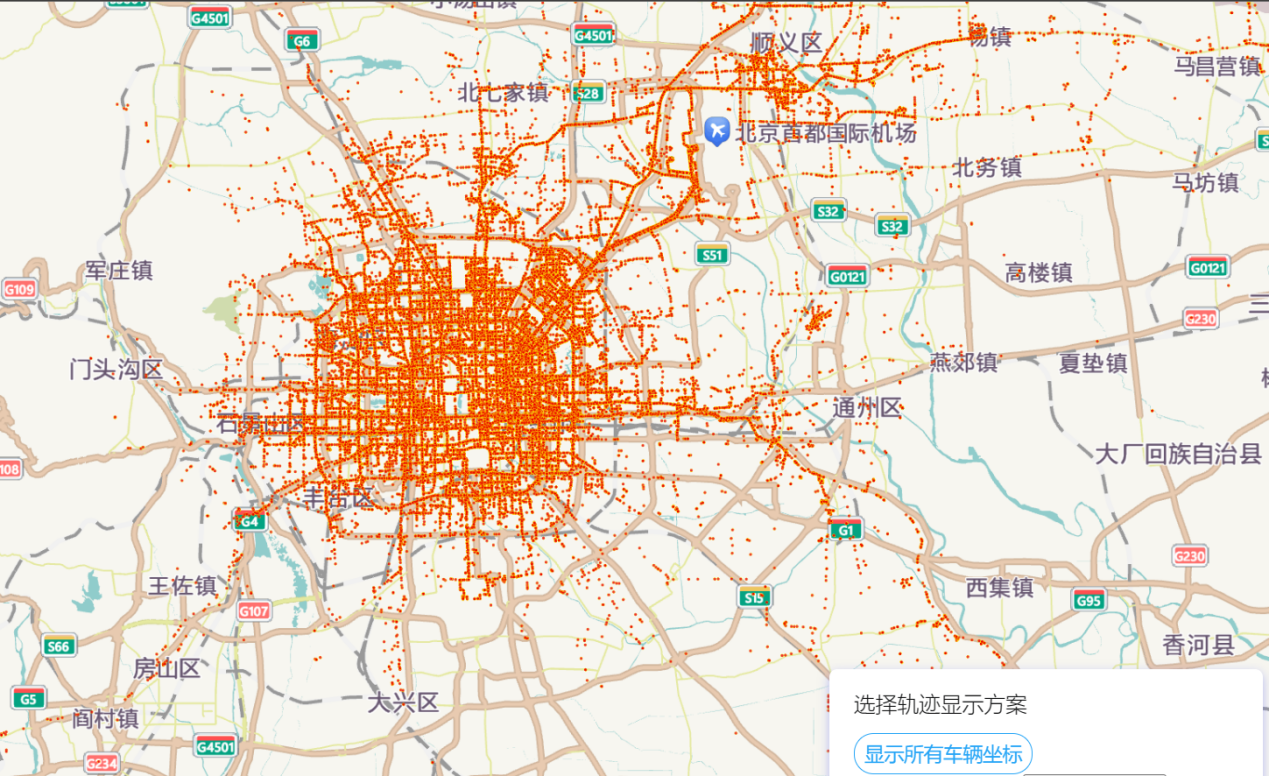


图 1.1.3“显示所有车辆轨迹”功能展示

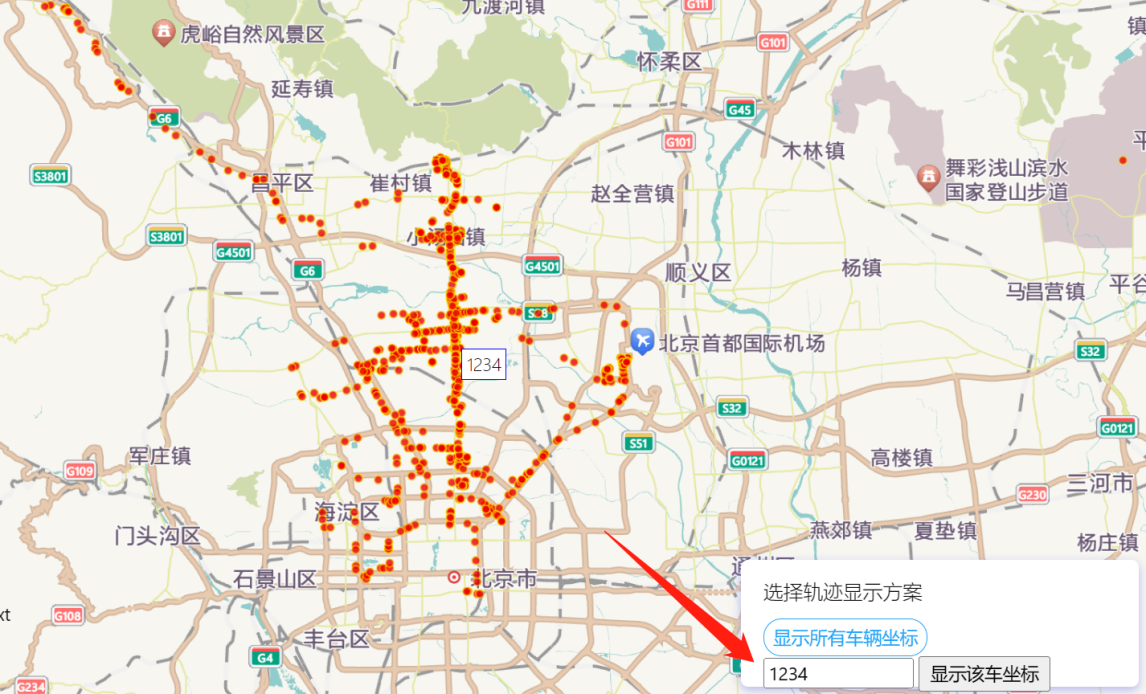


图 1.1.4“显示某一车辆轨迹”功能展示

1. **F2：地图缩放功能[公共部分]**

原始的高德地图API在电子地图导入时地图缩放功能默认开启，在此进行保留以完成F2.

1. **F3-F9用户交互界面**

**[公共部分]**

**总体思想：**为了实现JS网页端同C++程序端之间实现对接，通过不断查找资料，最终想到的方法如下：

1. C++代码完成后生成.exe可执行应用程序，以文本文件的形式作为输入和输出。
2. 前端JS创建用户输入界面，将用户输入的数据作为参数，以一定的格式保存到.txt文件当中。
3. 前端JS通过命令对.exe程序进行调用，后端exe读取所保存的.txt输入文件，然后根据所读取的内容，调用F3——F9与之对应的功能函数，传递参数，然后将运行的结果再以文件形式存放。
4. 前端JS读取后端exe执行后所生成的文件，将结果在网页端所展现出来。

通过以上思想将前端与后端联系起来，使整个项目变得更加完整、美观、紧密，更加有联系性。

用户输入界面：使用了勾选框的形式展现，用户通过勾选对应的文本框来选择执行相应的功能。

**2.功能实现[公共部分]**

**（1）前期数据预处理的类设计以及使用的全局数据定义:**

**taxi\_header\_use.h：全局头文件和变量文件,**

**extern\_data.cpp：全局变量和函数文件，**

**Point.cpp,taxi.cpp,position\_distribution\_inBeiJing.cpp,position\_distribution\_outBeiJing.cpp,time\_distribution.cpp,allpath.cpp,** **position\_distribution\_r.cpp,** **position\_distribution\_outr.cpp,均为类内声明函数的定义文件)：**

在轨迹分析中，数据的预处理是一大重点和难点，因此设计了多个类，定义在**taxi\_header\_use.h**中，其函数的定义在各自的cpp的文件中，以此来保证能够通过不同的形式分析、清洗和存储原始信息，使用对应的局部信息**。**

需要使用的全局信息同样以变量的形式在头文件**taxi\_header\_use.h**声明，在**extern\_data.cpp**中定义。

**[郑卓航]**

提前说明全处理过程中要使用的一组信息处理函数：

1. gettime函数，计算时间差。

参数共8个，分为两组的日、时、分、秒，两者相减并通过加权运算得到两者所差的时间，单位为秒。

时间复杂度很小，可以认为是θ(1)级别的操作，无空间复杂度。

1. Split函数，传入一个字符串和一个字符，对应的是一行数据，Split函数利用istringstream读取擦混入字符串，通过getline函数并以传入字符为依据将其分割为n段，n=传入字符出现次数+1，分割的每个子字符串作为vector<string>的一个元素添加到其中，把该容器返回。

时间复杂度很小, 假设传入一行字符串有k个分割字符,时间复杂度为θ(t+k)，t为字符串总长度但一般情况下k<=10，可认为也是θ(t)级别。

单次空间复杂度θ(t)，只是拆开了字符串，之后其他相关内容被全部丢弃了。

关于文件部分的说明：

1. 如果想使用文件读写序列化的版本，D盘要包括如下文件夹及文件：

D:\ taxitotal，内部有taxi1.txt-taxi10357.txt，存储了数据清洗后的全部轨迹点信息,另有taxitotal0.txt-taxitotal10.txt文件，用于出租车信息汇总

D:\ time\_taxidata，内部有2号-0时-0分.txt-8号23时59分.txt,存储了按时间分布的有效轨迹点信息。

D:\ pos\_taxidata，内部有两个文件夹，分别是inbeijing和outbeijing，inbeijing内部有115.400000经度,39.400000纬度.txt-115.400000经度,39.400000纬度.txt,存储了按空间分布的北京内有效轨迹点信息,outbeijing内部只有outbeijing.txt文件，存储北京外的全部有效轨迹点信息。

D:\pathdata，内部有pathdatagraph.txt文件，存储了图的全部点和边信息（约360万行），pathsortedge\_forcount\_class.txt，存储了按频繁度排序的路径边信息，pathsortedge\_forcount\_file，读写二进制文件，用于反序列化图的部分信息。另外有pathEdge和pathNode文件夹，内部存储的每个点和边信息，因为文件量太大不好打开以及名字的复杂性，只作为序列化的读取信息参考，不会参与任何后端操作。

1. 如果想使用boost读写序列化的版本，E盘要包括如下文件夹及文件：

E:\ taxitotal，内部有taxi1-taxi10357，存储了数据清洗后的全部轨迹点信息, ,另有taxitotal0 -taxitotal10文件，用于出租车信息汇总。

E:\ time\_taxidata，内部有2号-0时-0——8号23时59分文件,存储了按时间分布的有效轨迹点信息。

E:\ pos\_taxidata，内部有两个文件夹，分别是inbeijing和outbeijing，inbeijing内部有115.400000经度,39.400000纬度 -115.400000经度,39.400000纬度文件,存储了按空间分布的北京内有效轨迹点信息，outbeijing内部只有outbeijing文件，存储北京外的全部有效轨迹点信息。

E:\pathdata，内部有pathdatagraph.txt文件，存储了图的全部点和边信息（约360万行），pathsortedge\_forcount\_class.txt，存储了按频繁度排序的路径边信息，pathsortedge\_forcount\_file，读写二进制文件，用于反序列化图的部分信息。另外有pathEdge和pathNode文件夹，内部存储的每个点和边信息，因为文件量太大不好打开以及名字的复杂性，只作为序列化的读取信息参考，不会参与任何后端操作。

1. 必须在D盘有两个文件夹，一个是D:\result，用于获得函数执行后的写入文件结果。一个是D:\ big\_homework，用于前后端交互，前端写入 ，后端读取。

数据集的说明：

1. 数据集内有很多错误数据——定位不准确或者开关定位设备导致了轨迹点的误差，需要数据的清洗。
2. 数据集采样存在不合理性——有的车几秒一采样，有的车几分钟甚至几十分钟一采样，出租车的轨迹点数量并不平衡。
3. 数据集划分单一，面对多样的函数要求，需要进行多方面的预处理，否则时间和空间复杂度太大。

**[公共部分]**

针对不同出租车功能的需求，设计了如下几个类：

**Point类：**出租车的轨迹点类。能直观地将原始信息载入内存。

数据成员上，包括出租车的序号、时间（日、时、分、秒）、经纬度。

成员函数上，包括简单的赋值重载运算符（用于对轨迹点的唯一确定和排序），获得信息的函数和Point的初始化函数。

能够获得点的相关信息，完成对点的创建和复制。

**[郑卓航]**

必要的详细内容说明：

1. 数据成员taxiname因其最大为10357，所以指定为int。

times包括日、时、分、秒，用一个int数组存储，大小为4。

Longtitude：经度信息，用double存储。

Latitude：纬度信息，用double存储。

这时存储一个点的必要信息的最简化版本了。

因为经纬度在数据集精确到了小数点后5位，float在计算过程中会有误差，所以使用double。

1. Point的构造用上述信息的对应参数即可。
2. Point的init函数是为了第一次序列化使用的，这时候的传入参数是从数据集中得到的分割子字符串，对其处理和转化非常快，空间占用同Point类大小。
3. Point的init2函数是为了后面的文件读写使用的，比较频繁，但只有赋值操作，非常简单。
4. Point的get类型函数就是返回相关信息，简单迅速。
5. Point的输出函数不会被用户调用。
6. Point的赋值运算符是用来区分不同Point对象的，如果两个Point对象的taxiname和times不同时相同就是不同的点（时间唯一确定信息）。
7. 特殊内容：Point的比较操作：利用CMP类来实现（CMP.cpp）：

CMP类内部只有一个operator()函数，用于比较传入的两个Point，先比较其taxiname，如果相等就比较时间。这样就可以将Point对象有序排列了，顺序为序号名的在一起，大的在下面；同一序号名的时间晚的在下面。

**[公共部分]**

**taxi类：**将一辆出租车的Point聚合在一个类中，模拟一辆出租车的轨迹信息对

数据成员包括出租车序号、Point数量、指向（有效）Point数组的指针。

成员函数包括拷贝指针时必要的构造、拷贝函数和赋值运算符、析构函数，taxi的初始化函数，获得信息的函数以及输出信息函数。

能够获得出租车的相关信息，完成对出租车的创建、复制和相关信息输出。

**[郑卓航]**

必要的详细内容说明：

1. 数据成员正常定义。
2. 构造函数指定Point\*pv为空，表示出租车内部没有轨迹点信息，析构时强制pv为空，堆内存要返回给内存空间，避免浪费内存。
3. 拷贝构造和复制taxi时使用深拷贝，每次要在堆上申请空间和释放空间。
4. taxi根据数据集的初始化只有一次，以后都是利用文件传入读取信息或者反序列化进行初始化。这一次需要传入把数据集文件全部分割后的二维容器，每个一维元素对应一个有效轨迹点，遍历逐个初始化即可。时间复杂度不高（仅一次遍历），空间复杂度高（二维容器占据了很多内存）。
5. 但taxi的第一次初始化在函数中会进行数据的清洗：清洗方式为比较当前轨迹点和前一个轨迹点（第一个默认无问题），如果通过计算，两者之间的对应速度超过了67m/s，认为这个轨迹点是错误的或者无效的，不会用于taxi的初始化，经过第一次初始化的清洗后，后面所有相关操作都不需要考虑无效点了。
6. 获得信息和输出非常简单，但注意轨迹点的获得是根据其在出租车轨迹的次序获得的，一般不能单独取出，都是遍历获得。

**[公共部分]**

**time\_distribution类：**即**时间分布类**，以分钟为单位分割全部的出租车信息。用于表示某段时间全空间上的出租车和其轨迹点情况。

数据成员包括一个哈希表（无序）：

哈希表键值对：出租车序号—>出租车在该时段的所有Point类指针构成的容器（依照时间）。Point类指针内就有了出租车的序号信息和时空信息。

成员函数包括构造函数，初始化函数，获得信息函数和输出信息函数。

另外额外增加了初次序列化时对类内信息的更新函数和查询该时间范围的某个空间范围的出租车轨迹点信息函数。

能够获得并输出某一分钟的全部出租车相关信息，完成对某一分钟的信息初始化，能够分析出该时间内某个空间范围的出租车全部信息。

**position\_distribution\_inBeiJing类：**即**北京内的空间分布类,**以北京内的0.01经度或0.01的纬度变化划为一个类。

数据成员与时间分布类似，有一个哈希表，结构同上，另外多了该空间范围的起始位置经纬度信息。

成员函数包括构造析构函数，初始化+更新函数，输出信息函数，获得信息函数和查询该空间范围的某个时间范围的出租车轨迹点信息函数。

能够获取空间范围内某部（或全部）出租车的全部信息，可以完成某空间范围的信息初始化。按时间段查询。包括一个哈希表：出租车序号—该空间范围内全部点信息、能够分析出该空间内某个时间范围的出租车全部信息。

**position\_distribution\_outbeijing类：**即**北京外的空间分布类**，只有一个类对象作为一个整体使用。

数据成员同时间分部类，用一个哈希表存储键值对信息：出租车序号—出租车位于北京外的所有点信息（全时段）。

成员函数和用于基本同inBeijing类，但是没有查询空间范围内某时间范围的函数和功能（因为用不到）。

**[郑卓航]**

时空分布的三个类放在一起说明（逻辑相近）：

1. 数据成员均有类型unordered\_map，且其键为int，值为vector容器，元素为Point，这样做的好处是便于管理
2. 析构时，因为STL的特殊性质，需要显式释放以归还内存——vector利用clear和shrink\_to\_fit函数完成对占用内存的清理，unordermap则是和空的哈希表交换。
3. time\_distribution第一次的初始化是根据出租车类反序列化的结果进行的。每个类的更新函数需要传入出租车参数进行信息更新，另外因为时间分布类内部没有储存对应的时间信息，需要从外部传入参数判断什么时候是要分析的时间信息。

相对而言增加了时间和空间复杂度（每次都要传入一个出租车对象），但是优点在于可以单独对某个时间进行第一次初始化，避免了全部初始化时内存不够的问题。

position\_distribution\_outbeijing同理。而inbeijing的文件更新版本因为其要存储的对象众多，每次在外部先处理好出租车信息后获得要传入的轨迹点信息，每次只传入一个轨迹点完成信息的更新，以降低时间复杂度（序列化更新版本不需要）。

1. 输出信息和获得信息函数过程简单。
2. 查询某个时间、空间范围内的出租车轨迹点信息，三者均有此函数，用于后续的函数分析，因为保存信息的格式，此处的查询是无序查询，不能用算法化简，只能遍历选择合适的信息，所以时间复杂度相对较高，空间复杂度较小，只需要必要的信息即可。

结果保存出现的出租车序号数量和对应的相关轨迹点信息，虽然代价比起执只找数量高，但是可以作为其他分析的基础使用。

又因为对其他同类的对象查询是一个连续的过程，必须传入一个可变参数用于判断哪些出租车在其他同类查找被找到了，所以需要一个额外的bool数组作为判断依据，以略微提高空间复杂度为代价减少了每次查询都要根据旧信息获得相关出租车情况的时间代价。

因为涉及多参数返回，利用引用完成返回。

**[公共部分]**

针对特殊函数功能4的区域车流密度分析需要r区域分块的功能，设计了：

**position\_distribution\_r类：**只考虑北京内部：用于将r\*r个单位的格子的各段时间的信息存储到一起，1个单位等效于0.001经纬度，每段时间以10分钟为基础发生变化。

数据成员包含：静态变量r，用于记录单位数量；静态变量vsize，记录一共要分析多少段时间变化，两个vector容器，一个内部元素仍为vector容器，用于在二维层次上记录每个格子出现的出租车的数量；另一个内部元素为set（对应红黑树），用于记录每个格子出现的轨迹点信息，轨迹点进行排序，优先级为为序号名权值、时间信息权值。

成员函数包括：获得信息函数，构造析构函数，插入点信息函数。

用于分析r\*r时每个格子的详细信息。

**position\_distribution\_outr类:**将北京外的区域作为一个整体格子进行划分。

数据成员包含了：记录出租车是否出现过（bool数组）、记录每时段出现的出租车次数（单个车只计算一次，vector数组），以及北京外每个时段出现的全部出租车轨迹信息（vector套set）。

成员函数包含了构造和析构函数，获得信息函数和插入点信息函数。

**[郑卓航]**

1. 之所以以类的形式进行保存，是因为r\*r区域的格式和空间分部类十分相似，一些代码可移植，另外是为了保存形式更合理、更有序。

同时，在position\_distribution\_r内部可以直接用静态成员变量进行对不同类的统一联系，每次函数调用导致的静态数据变化会区别开不同的函数调用，而单次函数调用又能够保证数据的连续统一。

1. 为了减少其占用的空间，其内部的轨迹点容器能空则空，插入点信息时不会根据时间段数进行容器的初始化，而是根据最后一个有数据的时间段数进行初始化。

比如分析100分钟的变化，每段10分钟，共10段，如果有4段时间该区域无信息，最后一个有数据的区域是8，那么轨迹点容器数量为8个，且有两个轨迹点容器内部是空的，也就是只会对有数据的区域进行保存。从北京的整体角度分析，因为有很多区域都是空的，这样的数据处理极大的降低了空间占用。

例如同样执行高峰时期的区域划分函数，每个容器均初始化，内存占用达到了惊人的3.2G，而这种部分初始化的方式只占用了800M。

1. 析构时同样需要将内存显式释放。
2. 获得信息时比其他类的获得信息要复杂，因为容器数量可能小于时间段数，直接访问会导致崩溃，所以访问或遍历要根据容器的数量和大小决定遍历情况，导致了时间复杂度的增加和代码量的增加。

如果想优化，可以通过链表的形式进行存储。（代码代价太大，未优化）

1. 边界条件的处理不细考虑，因为划分网络主要是为了北京内部，边缘地区数据量非常少，可以忽略，所以整体上的区域分析不会出现问题。

**[公共部分]**

针对路径分析设计了：

**pathNode类：**即**路径节点类**，用于存储对应路径的大致被划分到空间的信息，作为以下两个类的基础类。

数据成员只包含两个信息：经度和纬度，精确到0.001。

成员函数包括获得节点经纬度和构造函数。

**pathEdge类：**即**路径边类**，用于存储相邻路径的空间信息。

数据成员包含以下信息：边的两个节点（无向）、边的频繁度（即示该边被经过的次数），边的物理长度（由两个节点的经纬度计算得到）、经过该边所花费的最短时间（当存在多次经过时）、达到最短时间所用的起始点和终点（有向）。

成员函数包括构造函数，信息更新函数（传入一个点后的变化），信息获得函数，边的信息初始化函数，信息输出函数。

用于分析某个路径的相关性质和信息。

**pathgraph类：**即**路径图类，**用于存储所有节点和边，构成一个图。

数据成员包含两个哈希表，其中一个用于存储全部出租车的结点信息，并通过两个结点位置生成字符串作为键值来对应结点形成的类信息；另外一个存储边：通过无向边生成字符串作为键值对应边形成的类信息。

成员函数包括构造、析构函数，更新函数和初始化函数，信息获得函数，信息输出函数，以及序列化时要使用的函数（下面详细说明）。

**[郑卓航、祝锦红]**

1. 路径图类的缺点是Edge类的存储空间代价大，但是优点是维护后进行其他操作的时间代价非常小。
2. 边的文件版本的初始化参数复杂，但实际上时空复杂度非常小。
3. 结点和边的获得信息函数和输出函数简单迅速。
4. pathgraph的函数虽然很多很复杂，但实际上基本只用于预处理的部分，比如传入出租车对图的更新和序列化函数，执行一次后就不会再执行了，用户不需要关心此部分，预处理完成，再调用排序函数排好序后写入文件，就能够直接使用成果了，非常简单。时空复杂度非常小（大的部分全在预处理）。
5. 函数功能6需要使用其函数进行局部图的构建，是全部函数功能中执行时间和空间代价最大的一个操作，毕竟需要对矩形内的信息进行路径和图的构建。
6. 获得信息函数和输出函数非常简单迅速。

**[公共部分]**

全局变量相关信息：是接下来所有操作中频繁的相关信息。

1. 经纬度之比：对应北京地区1经度和1纬度变化的距离之比：0.766044443
2. 北京的最大最小经纬度：115.40, 117.60, 39.40, 41.10
3. 分析的出租车数量:10357
4. r\*r对应的基础经纬度单位：0.001。

**[郑卓航]**

1. 因为全局变量贯穿全后端的编写，因此在头文件中声明使用，但不会破坏封装。
2. 不会修改全局变量。

**[公共部分]**

**（2）序列化和数据清洗（taxi\_init.h，taxi\_init\_data.cpp）**

建立了类后，可以将原始数据转换为各种各样的数据，为了更好、更方便地存储和利用这些数据，为每个类实现了序列化和反序列化的函数。

序列化和反序列化基于C++boost工业库中的archive二进制相关库（作为读写的类型依据）以及serialization相关库（作为处理STL相关内容的依据）。

另外，为了不同成员运行时能够更好的兼容相关内容，除了基于boost库的序列化外，也基于C++标准库的文件写入写出的相关库和函数进行了信息存储和转存，这样就有了两个版本的序列化和反序列化，一个是boost的二进制，一个是文件的读写。

在转换数据时，通过分析发现出租车轨迹点有很多点不符合实际生活的正常轨迹，可能原因是定位的偏差或者数据集统计信息时发生了错误。

面对这些问题，在序列化时需要对每个原始的数据集文件中的错误数据进行数据清洗，让轨迹点更加合理，不会出现诡异的错误。

**序列化前的准备：在所有涉及到的类中声明友元模版函数serialize，传入所有相关成员从而完成序列化和反序列化时的信息传入。**

**[郑卓航]**

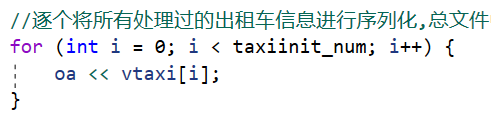
1. 序列化和反序列化比起文件读写速度快了接近一倍，而且二进制版本的空间占用比txt版本的更小，更精确，而且不会被指针影响，二进制读写的方式能够保证信息的准确性和隐蔽性。
2. txt版本可以使用户或者开发者直接阅读信息检验是否有错误，而且可移植性强，不会因为版本问题失效。
3. 数据清洗是有必要的，否则会有从北京到北极再回到北京这种超远距离的飘逸出现，严重影响分析和可视化。
4. 序列化的相关操作较为繁琐，这里不赘述，但是注意指针的序列化和反序列化要循环进行；STL容器的序列化和反序列化需要引用boost库的一些特殊头文件完成对应操作。

**[公共部分]**

**taxi类序列化：**

序列化前面的过程主要在于读取原始数据、按逗号分割数据完成形式的转存并将这些转存数据按taxi类的vtaxi数组依次进行初始化。

序列化和文件写入的核心是将类的信息转换为能够序列化和写入的信息。序列化利用文本的输出归档类boost::archive::text\_oarchive oa完成，文件写入利用ofstream完成。

最后将信息输出到boost::archive的序列化流中或者ofstream的相关写入文件流中，得到序列化的本地文件。

taxi类文件的序列化或者写入均以出租车序号为单位，每个出租车对应一个文件，这样之后的操作就不需要一次性全部反序列化或者读入了。

（实际上，taxi还有一组taxitotal文件，每个文件存入了1000个出租车，本意是用于全面分析使用，后面发现无用，就弃置了）。

在这个过程中，对数据进行检查，如有错误数据则消除。

**taxi类反序列化：**

针对一辆出租车，要对其反序列化，可以采用类似的方法，将ofstream改为ifstream即可。

多个出租车只需要传入对应的序号即可全部反序列化。

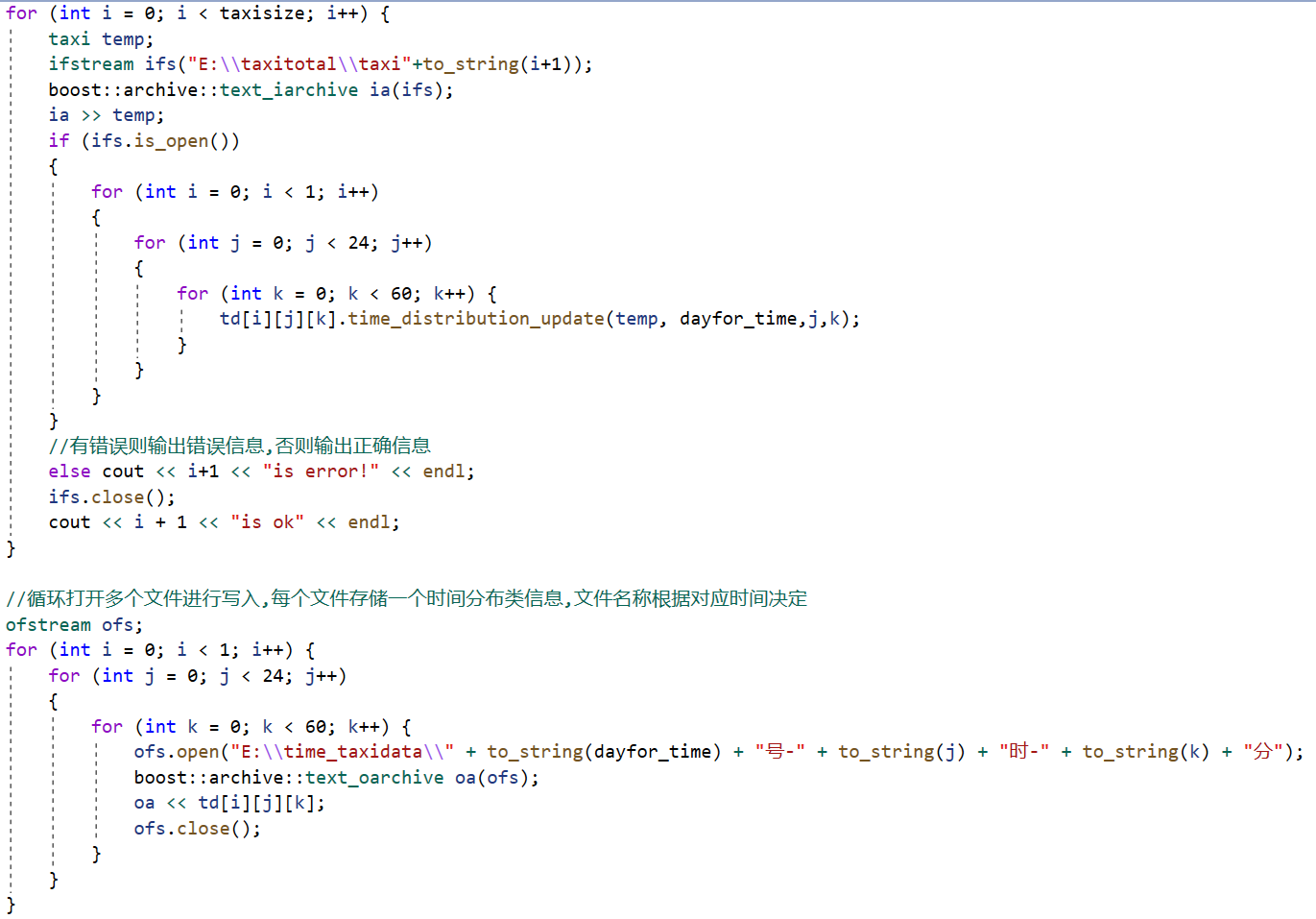


文件的读取方式则在taxi类和Point中定义好读取函数，利用读文件的按行按逗号分割分割信息，再挨个读取（逆写入操作）。

**time\_distribution类序列化：**

该类的序列化需要出租车类序列化和反序列化支持。

假设已经将出租车信息写入了本地序列化文件，那么可以将其反序列化出来，然后按照time\_distribution的格式将数据写入到time\_distribution的类对象数组中，最后以同样的方式（ofstream和boost::archive::text\_oarchive）将对象信息序列化到本地文件中。



文件的写入逻辑同上，只需要用文件的对应操作即可。

时间分部类的文件单位是1分钟，也就是一共有7\*24\*60个文件，每个文件对应了一分钟的详细信息。

**time\_distribution反序列化：**

传入时间参数（日、时、分）后，用类似的方式（ofstream改ifstream）反序列化time\_distribution文件到time\_distribution对象。

文件的读取逻辑同上，只需要用文件的对应操作即可。

空间分布类和pathgraph类也实现了类似的序列化和反序列化，文件的读取和写入操作。

空间分布类的文件单位是0.01经纬度的变化，加上规定的北京经纬度范围，可知共有220\*170个空间分布类的文件，外加一个北京外的文件。

pathgraph的文件只有一个，将其信息全部存入其中，但也为每个结点和路径进行了序列化(文件数量巨大，点以十万为单位，边以百万为单位)，这是用来观察的，基本不会调用。

**[郑卓航]**

1. 文件读写的版本的操作遵循一套流程：

写入上，逐行写入，每行用逗号分割有效信息，输入时注意经纬度需要精确到小数点后五位，输入文件ofsteam流完成即可。文件名根据函数执行传入的参数决定，比如时间分布类的话文件名会包括日、时、分信息以帮助确认是否是要打开的文件。

读取上，逐行读取文件，每行信息用Split函数进行对逗号的分割获得子字符串，将子字符串进行类型转化获得对应的类型信息传入进行类对象的初始化即可。

1. boost序列化和反序列化的操作遵循一套流程：利用模版友元函数侵入式完成对象信息的二进制序列化和反序列化，相关的序列化和反序列化对象绑定到文件流对象上进行相关操作即可，简单方便快捷。
2. 所有操作全部封装为函数，方便随时使用和移植。

**[公共部分]**

**（3）功能函数设计(taxi\_header\_use.h,function1.cpp,function2.cpp)**

**任务一：F3：区域范围查找**

实现函数：int function\_findtaxinum\_somewhere(set<Point, CMP>& s,int day, int hour, int minute, int time\_span , bool output)

set<Point, CMP>& s：存储区域内点信息，内部按车名和时间排序

double\* pos：4个元素，矩形西经度、东经度、南纬度、北纬度

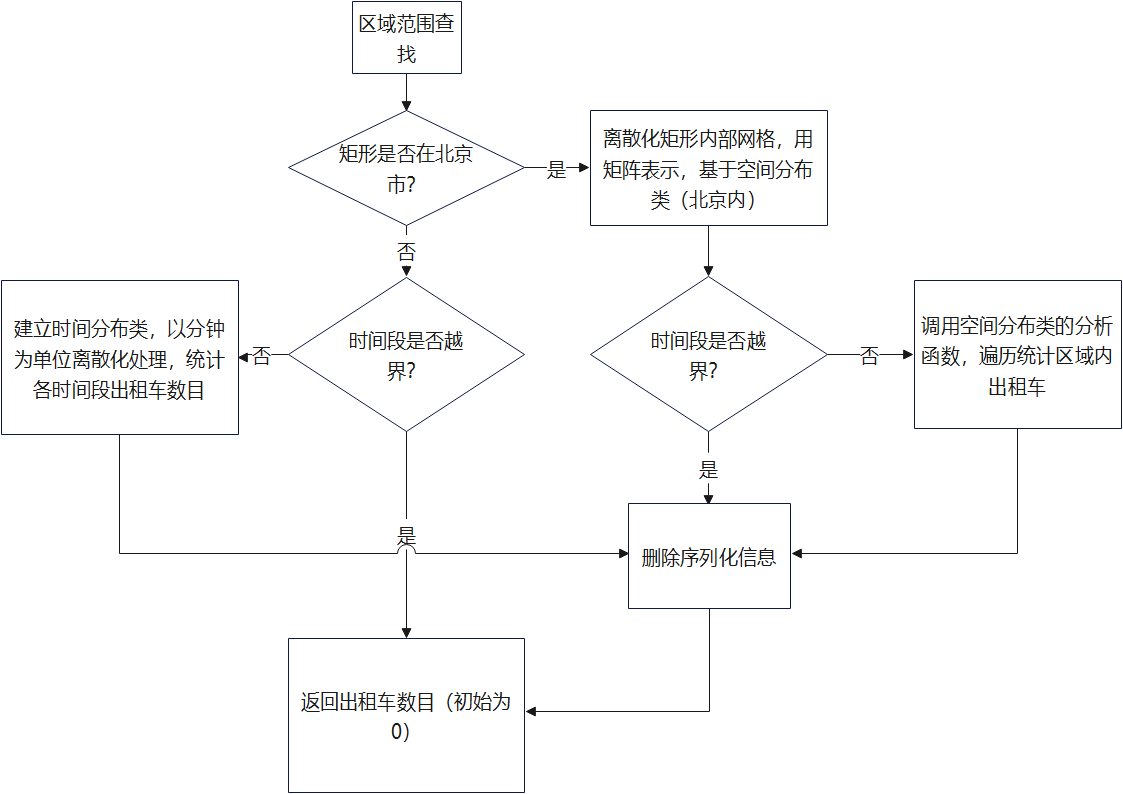
int day, hour, minute：初始时间

int time\_span：时间跨度（步长）

bool output：是否输出过程结果到文件中

返回值：返回矩形区域内出租车数目

具体流程如下：



删除序列化信息之前会将统计结果输入文件中，第一行是出租车数量，后面是每个出现的出租车的序号和轨迹点信息，此文件被用于前端读取和输出。

此函数因为能够统计整个要求矩形的信息，除了完成本身功能外，还可以作为其他功能的前置函数使用。

优化：当时间段越界后会调用到最后的时间信息处，统计出租车出现信息不会直接去删除序列化信息，初始时间越界不变。(最后时间信息为8号23点59分)

**[郑卓航]**

1. function\_findtaxinum\_somewhere一般情况下返回出租车数量，第一个参数存储相关的轨迹点信息，这个可根据情况使用。
2. 内部判断信息合理性，不合理的话会直接写入文件无效信息，或者无相关数据结果（比如这个矩形真的一直没有车经过），非常易于起前端判断是否成功。
3. 合理利用floor、ceil和round函数去明确保存小数点后几位并消除机器无法准确保存浮点数的误差。
4. 矩形有北京外的经纬度信息就从时间角度初始化是因为边界条件太复杂，以及用户指定的随意性从全球角度考虑太复杂，另外这种情况不会很常用，所以这样设定。能够有效降低时空复杂度和大幅减少代码量，更可读。
5. 虽然函数内很多地方都是遍历居多，没有特地设计算法，但是复杂度依旧不高，和序列化、反序列化的复杂度级别相近，不会导致程序运行缓慢。
6. 如果想要进一步大幅提高运行速度，必须建立数据库，但代码代价太高，再加上本次大作业为多人合作完成，所以放弃了数据库的建立方式。
7. 该函数因为提供了额外信息导致了复杂度一定程度的提高，但是为后续的函数编写提供了良好的基础，相对而言，比较方便。

**任务二：F4：区域车流密度分析**

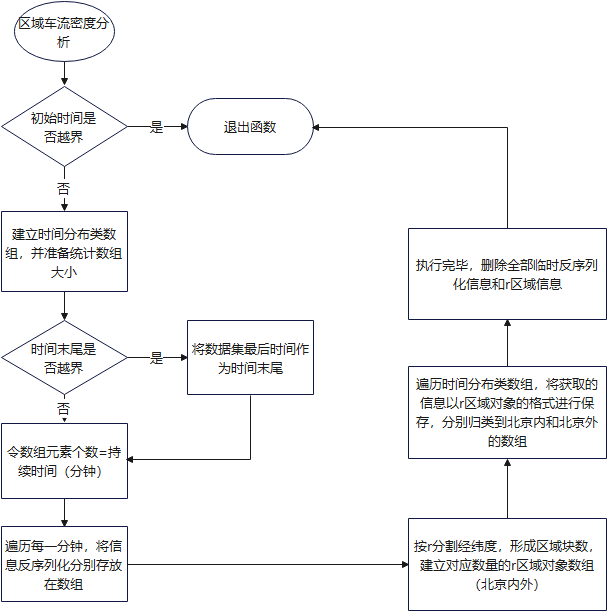
在设计中，一段时间的某个区域（格子）的“车流密度”以出租车数量为标准，车流密度的变化就是在两段时间上这个区域出租车数量的变化情况。

实现函数：void function\_regional\_density\_analysis(int r, int day, int hour, int minute, int time\_span, bool output)

r:距离参数单位数量，如r=10对应0.01\*0.01经纬度大小作为一个格子。

其他参数意义和上方的相同。

具体流程如下：



运行结果在中间过程（“执行完毕前”）输出到文件中，函数本身不返回结果。

**[郑卓航]**

1. function\_regional\_density\_analysis的缺点很明显，用户指定的距离参数不是一般的公制单位，而是0.001经纬度单位的系数，对用户的使用有一定影响，但是受限于代码复杂性和电脑性能，只能这样处理。
2. 以10分钟为单位分析变化是因为考虑到数据集采样的不稳定性，10分钟是一个比较合理的数字，实际上10-30分钟均可，这里用10分钟也是为了演示变化。
3. r\*r格子的划分是综合了时空代价的选择，最大化了二者的平衡。
4. 之所以不直接使用空间分布类进行分析，是因为其跨度太大，很难观察到更小地区的变化，而如果修改空间分布类，不利于其他函数的调用以及信息的存储。
5. 该函数的返回结果同样有获取信息的冗余，除了数据变化外，还能看到是哪些轨迹点和出租车的变化，这是为了后期检查是否有数据集采样的影响，实际证明影响并不算大。

**[公共部分]**

**任务三：F5：区域关联分析1**

这个任务需要统计不同时间段往来两个矩形区域的车流量的变化。在这里定义：当同一辆车在两个矩形区域均出现时，对往来这两个区域之间的车流量贡献增加。反之，如果有车在其中一个区域出现但另一个区域没出现，或者在两个区域都没出现，就不算车流量。

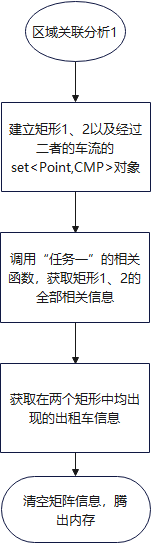
实现函数：set<Point, CMP> function\_region\_association\_analysis\_one(double\* pos1, double\* pos2, int day, int hour, int minute, int time\_span, bool output)

double\* pos1, pos2 ：两个矩形的方位

其他同上

返回值：点集，可以反映车流量

流程如下：



获取信息时，要注意此信息不包括不在两个区域内的任意轨迹点，不会显示一条条完整的路径。

信息结果同样写入到文件中。

**[郑卓航]**

function\_region\_association\_analysis\_one基本基于函数功能1（F3）完成，后面的处理其实并不复杂，空间复杂度较小，因为基本不存在多次嵌套遍历，时间复杂度也并不高，基本在θ(n)级别，n为加入到两个矩形中的轨迹点数量（不是出租车数量，因为要分析轨迹点）。

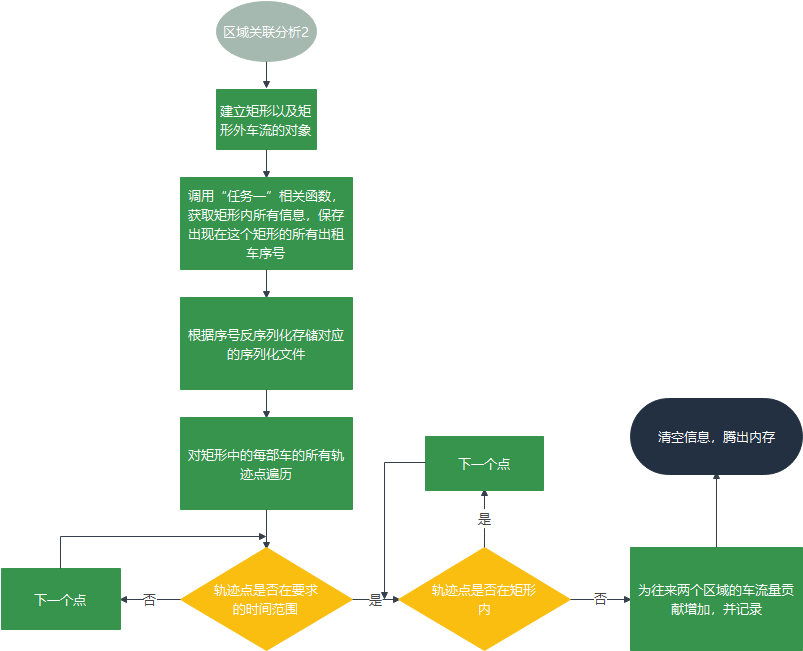
**[公共部分]**

**任务四：F6：区域关联分析2**

这个任务和F5的区别在于：其中一个矩形区域变成了另一个矩形区域的“补集”，即以外的所有地区，因此处理方式相差不大。

实现函数：void function\_region\_association\_analysis\_two(double\* pos, int day, int hour, int minute, int time\_span, bool output)

在此给出函数流程图：



注意，不会出现一个出租车查不到在矩形内的点（不然的话不会出现在矩形的信息中）。

结果写入文件中。

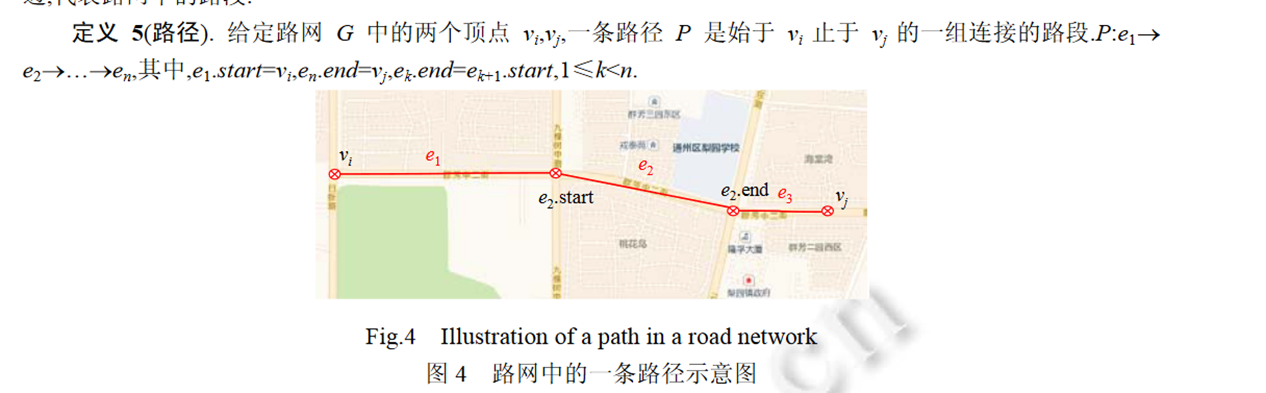
**[郑卓航]**

1. function\_region\_association\_analysis\_two同样基本基于函数功能1（F3）完成，后面的处理比起前一个稍微复杂，因为需要检测在矩形内的出租车的全部轨迹点判断其与其他区域的关联情况，需要纳入考虑的轨迹点更多，因此时间复杂度比起前者更低（只需要判断一个矩阵的情况，检测一遍涉及的出租车），空间复杂度更高。
2. 如果矩形内的出租车非常多，会导致时空复杂度的迅速上升，同样想要减少此情况的出现只能建立数据库。

**[公共部分]**

**任务五：F7：频繁路径分析1**

这个任务的需求描述并不直观，在此定义两个量。其一是“路径”，采用“路网匹配算法综述”（高文超等）的定义，如下：



注意路径是有向边两两首尾相连的结果。

其二是“频繁度”，定义：一条路径的频繁度是这条路径上通行的汽车的总数。

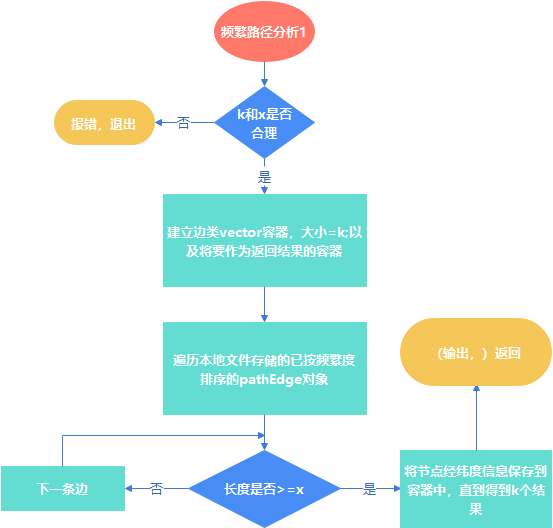
该功能因为较为特殊，是对全局信息的处理，所以进行了复杂的预处理，获得了按频繁度排序的路径边文件，这样就不必每次运行都进行相同的操作了。

实现函数：vector<vector<string>> function\_path\_frequent\_analysis1(double x,int k, bool output)

x对应的要求的最小长度，k对应要寻找的路径数量。

返回值：第一层vector:“边”；第二层vector：四个元素，表示两个节点的经纬度

流程如下：



时间复杂度极低，只需要遍历一次文件内容即可。

结果写入到文件中。

**[祝锦红]**

1. function\_path\_frequent\_analysis1本身执行代价基本为θ(n),n为路径数，最极端的情况下也只需要遍历一遍即可，绝大多数情况下只需要θ(1)的时间完成。
2. 预处理的逻辑如下：

逐个出租车输入到图中进行结点和边的更新，完成后反序列化；

随后对这个图的信息进行处理，因为不能对无序哈希表直接排序，利用vector容器将全部路径存入其中（空间代价大，但是只需要执行一次，不影响用户），并对其按频繁度排序（利用拉姆达表达式快速完成比较函数），随后输入到文件中，这个结果文件就保存了该函数功能所需的全部信息了。

1. 以空间换时间。

**[公共部分]**

**任务六：F8：频繁路径分析2**

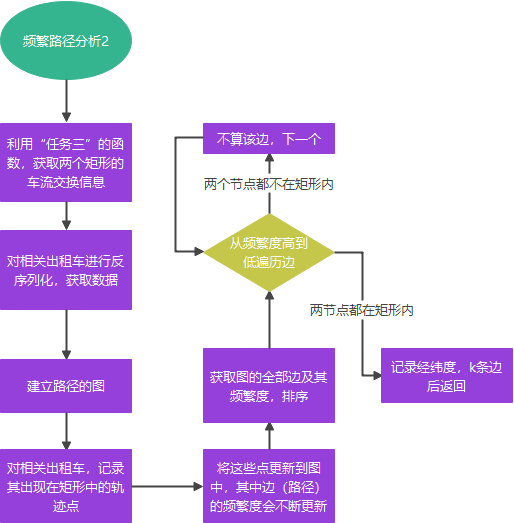
这个任务涉及到两个矩形，也涉及到前面关于频繁路径的实现。因此将二者结合来实现。

实现函数：vector<vector<double>> function\_path\_frequent\_analysis2(double\* pos1, double\* pos2, int k, bool output)

参数含义均同上。

返回值：节点经纬度信息数组

过程：



前面的任务中对频繁度进行了预处理，这里因为限定了区域，不能直接读取频繁度文件，只能通过一步步建图来更新路径的频繁度。

考虑到k条边，实际上可以将排序的步骤改为建立大顶堆来存频繁度，取前k个堆顶元素即可。但实际上都需要去获取图的全部边（得到全部边的前提操作占用了最多的时间，后面的反而不重要了），因此时间复杂度区别不大，为了让程序更“显而易见”，采用排序的方法。

另外设计了仅仅分析相邻点的路径情况，这时，只需要考虑依次读取的边的两点分别在两个矩形中（外部存储时已经按频繁度排序），读取k条即可。

**[祝锦红]**

1. function\_path\_frequent\_analysis2的执行复杂度最高，代价最大，因为需要建立一个局部图，路径的设定对时间和空间的影响太大。
2. 获得通过局部的车流交换信息可以最快速的方式获得有效信息。
3. 该函数综合运用了前面的函数功能、各种类的数据和成员函数，代码代价小，功能移植性强。

**[公共部分]**

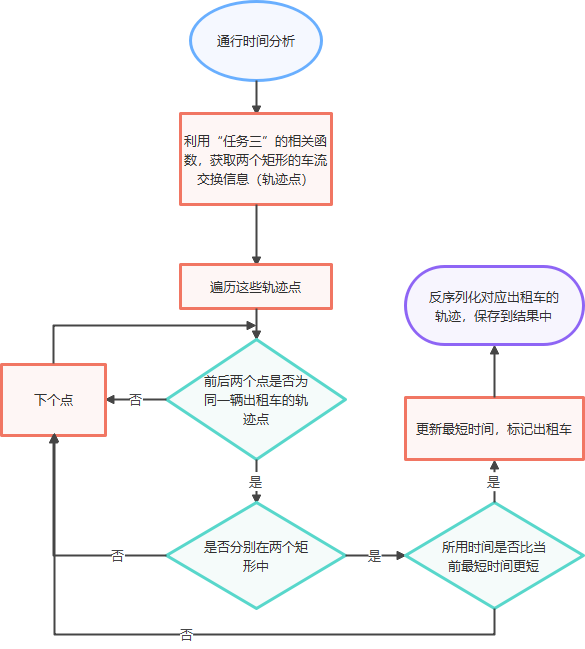
**任务七：F9：通信时间分析**

实现函数：vector<Point> function\_pathuse\_shortesttime(double\* pos1, double\* pos2, int day, int hour, int minute, int time\_span, bool output)

参数含义同上。

返回值：一系列点，依次相连后得到最短通行时间的路径

实现流程如下：



注意，反序出租车轨迹时起始点就是标记的起点，终点就是标记的终点，因为中间可能存在不在两个矩形内的轨迹点，所以要用最后结果的出租车序号反序列化得到这些不在矩形内的点形成最短时间的轨迹。

**[祝锦红]**

1. function\_pathuse\_shortesttime的核心逻辑和上方类似，都是通过快速获得交流信息完成后续信息处理，因此此处只需要寻找最短时间路径，所以时间和空间代价都很小。
2. 该起点和终点对应的路径信息在大数量的情况基本不会产生误差，但是如果矩形内车辆稀疏的话可能并不是合理路径（比如是从A高速到C再到B，这样就不如直接的A到B），但由于数据集的局限只能完成此程度。

**[公共部分]**

**（4）将后端编写内容全部打包到一个可执行程序（exe）中（taxi\_exe，Taxi\_trajectory\_analysis.cpp）：**

因为前端使用的需要，完成了对后端文件的写入和代码的编写后，使用vs2019中的插件Microsoft Visual Studio Installer Projects将整个项目打包生成一个exe文件用于执行对应函数。

函数的执行根据读取前端提供的文件进行分析要执行哪个函数，参数有哪些。

主函数的定义：

打开前端提供的文件，读取第一行，仅一个数字，范围为3-9，分别对应了任务书功能要求的F3-F9；读取第二行，用逗号分割信息，与要执行的功能函数的参数一一匹配。

整体先读取两行，信息存入vector容器中，再用一个switch实现第一行判断进行哪个功能函数，再将读取的相关参数经类型转化后传递给函数执行即可(生成的文件包括了运行结果，可被前端读取使用)。

**[郑卓航]**

1. exe的版本是debug版本，是因为序列化的调用只能通过debug版本完成，美中不足。
2. 实际上生成的是一个.msi版本的安装包，可以安装到windows中，相当于一个小型的应用，可移植性非常强，缺点是不能打包数据，数据和执行文件是分离的。
3. 该exe可以作为前端的一个调用文件，是前后端交互的基础（另一个是文件读写）。

**[公共部分]**

三**、软件测试**

后端部分不需要前端的软件测试，只能测试exe文件。

经测试，在信息合理的情况下，exe能够成功运行，并在D:\result文件夹下生成了合理的结果文件（详见resultexample文件内文件示例）。

**四、总结**

在本次数据结构课程设计中，我们选择了出租车轨迹分析——倾向考察图论知识点的任务。与一般的图/网络分析不同的一点在于，我们需要将现实中的路网——一种连续的、复杂的网络，转化为容易被计算机处理以及方便用户体验的离散化模型。因此我们在前期做了大量的数据预处理工作，这方面还包括剔除异常点，设计各式各样的类来加载数据。最后将功能归类到一个一个函数中。

同样重要的是对用户需求和任务结果进行可视化，这方面需要查阅大量资料，同时要有比较扎实的JavaScript语言基础。在付诸实践的时候，需要清楚关于回调、点击事件的知识，以及掌握JS API的功能和操作方法。最后，还要设计网页的样貌，这方面就要借助HTML和CSS的知识。

我们设计的程序基本完成了任务，但是还存在诸多问题。一个是所应用的数据结构并不明显。在后端C++程序中，我们采用了大量的嵌套的unordered\_map、set等体现哈希表和红黑树的数据结构，这是很容易看出来的。但是我们没有显式地去构造更多的数据结构，而是采用了离散化和转换为矩阵的方法将任务本身变得容易让计算机处理，或者是隐含了大量数据结构。此外在设计过程中，对于C++和JavaScript的衔接，计划包含了采用Emscripten的WebAssembly或者Embind去设计胶水代码嵌入JavaScript和C++模块，让JS端能直接构建和调用C++类和对象，但是考虑到复杂性，以及需要额外在JS端和C++端添加大量代码，最终舍弃了这种实现方式。

一言以蔽之，此次课程设计完成了任务，但仍需要更多的打磨和改进，希望在以后的项目设计中能做得更好。

**[郑卓航]**

个人总结：后端的实现内容非常复杂繁多，很多地方没有经过算法的优化，但完成度较高，能够将对应的函数功能合理实现（如果有服务器的话能实现的更好），能够合理运行数据结构的知识选择合理的数据结构完成数据信息的存储和封装，也较为完美的实现了c++多文件的分工，使代码整体更清晰可读易懂。

如果想要进一步提升，除了设计优秀的算法，使用更合理的数据结构外，还可以参考数据库的实现完成更合理的数据处理和分析。

**[祝锦红]**

个人总结：我主要担任项目整体设计、函数功能F7~F9的开发，以及PPT和课程设计报告公共部分的设计。函数功能的开发延续了郑卓航的风格，在原有的类的基础上做了扩展，算法上考虑了很多，最终选择了一些比较“朴实”的算法，在涉及到文件读写的部分做了很多工作。

但其实也有很多时间用于计划中的C++与JavaScript的衔接部分。由于个人水平不足以及程序的复杂性，使用Emscripten编写的胶水代码（程序）在开始可以胜任衔接的工作，但随着任务的推进，包括外部库和本地文件操作。胶水代码有其局限性，并不能将整个项目的内容囊括其中，最终整体放弃了这部分内容，这也是本次设计的一大缺憾。

通过本次课程设计，学习到了更深层次的数据结构和算法的内容，也得到了项目编写和统筹的经验，也期待以后的项目设计能吸取教训，在一次次学习中得到成长。

**[成云]**

个人总结：在本次实验中，主要负责的是项目的前端工作，以及函数F1、F2的实现。F1、F2直接在html中实现，因为数据格式原因和html本身对于读取用户本地文件的限制，只能将所有的出租车的数据格式用c++更改成js中的数组格式，才能实现html对本地txt中数据的读取与处理，在完成这一步之后，成功利用高德地图API实现了打印出租车轨迹。

在其他函数的可视化部分遇到了一些问题，可视化和函数功能之间的接口处理也遇到了问题。因为html无法直接读取本地txt文件里面的数据，所以只能通过用户手动下载、手动选择来实现保存用户输入的数据和展示运行的结果。在这个问题上，我和柴豪研究了很久都没有找到方法来解决，留下了一个遗憾，通过服务器作为中转站可以解决这个问题，但是无法兼容以及实现的函数功能，最后也不得不放弃了这个想法。

在本次实验中，了解到了更多关于前后端的知识，学习到了项目实现的过程，积累了项目实现的经验，期待以后在遇到问题和解决问题中得到成长。

**[柴豪]**

个人总结：我主要承担前端部分的设计和实现，包括功能函数F1和F2的实现，以及前端部分与后端部分实现对接，通过与组员之间的讨论最终采取在前端通过JS调用本地应用程序的方法实现前后端之间的联系。在本次课程设计实践中，从一开始只是使用API完成功能F1F2，到后来决定将整个项目所有功能放到JS网页展示，其间所消耗了大量时间去学习和运用HTML网页以及JS代码的编写。整个过程得到了很多收获，掌握了许多前端部分的知识；同时，也遇到了许多服务器端与本地文件之间的联系上的问题，没有找到比较完美、准确的方法来实现功能。虽然有一点小失败，但是通过这样的尝试，也让我掌握了许多此方面的知识储备，也是为将来做前端开发项目积攒了一次宝贵经验。

**五、参考文献**

[1]高文超,李国良,塔娜.路网匹配算法综述[J].软件学报,2018,29(02):225-250.DOI:10.13328/j.cnki.jos.005424.

[2] Boost C++ LIBRARIES. Serialization[EB/OL]. [2023-5-12]. <https://www.boost.org/doc/libs/1_82_0/libs/serialization/doc/archives.html>.

[3] 高德地图开放平台. 地图 JS API 2.0[EB/OL]. [2023-5-12]. <https://lbs.amap.com/api/jsapi-v2/summary/>.

[4] [【大数据课程设计】出租车轨迹数据分析\_野犬1998的博客-CSDN博客](https://blog.csdn.net/qq_42319367/article/details/106951043)(仅参考其逻辑) [EB/OL]. [2023-5-11]

[5] [VS2019项目打包exe文件教程\_vs2019如何打包exe文件\_BIGBOSSyifi的博客-CSDN博客](https://blog.csdn.net/weixin_50679163/article/details/119702995). [EB/OL]. [2023-5-11]

[6] T-Drive trajectory data sample [EB/OL]. [2023-5-12]. [T-Drive trajectory data sample - Microsoft Research](https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/)

[7] Emscripten Documentation [EB/OL]. [2023-5-12]. <https://emscripten.org/index.html>