2018 年 12 月 19 日

PROJECT TWO'S DOCUMENT

1. BRIEF INTRODUCTION

对本次 project 进行简单介绍。

A. 引入了一个名为 JXL.jar 的 java 包，主要用于为读取 Excel 文件提供函数，帮助读取 Excel 文件（在目录 WebContent/WEB-INF/lib 下）。

B. 本次 project 修改或创建的 java 文件都在 src/com.fudan.sw.dsa.project2 目录下。

i. Bean 目录下的内容：Address.java ReturnValue.java（前面的两个 java 文件是助教所提供的，没进行任何修改。后面的所有 java 文件都是由我独立编写完成）Dijkstra.java Edge.java Graph.java interchange.java Vertex.java。

1. Dijkstra.java：用于执行 Dijkstra 算法，其中使用了 java 本身自带的优先队列 PriorityQueue 来辅助执行 Dijkstra 算法。
2. Edge.java：用于构建图的边实体，对每个 Vertex 而言，与它相关的所有的 edge 都存放在 Vertex 的 neighbours（ArrayList<Edge>）中
3. Graph.java：用于构建图的实体，包含一些基础的构建图的函数
4. Interchange.java：用于实现 project 中所期望的最小换乘的问题，其中使用了 dijkstra 算法来实现查询最短线路（用地铁线路构建的图）
5. Vertex.java：用于构建顶点的实体，包含一些基础的属性与函数

ii. Config 目录下的内容：RootConfig.java WebApplnitializer.java WebConfig.java。（都是由助教提供，没有进行修改，因此不做介绍了）

iii. Constant 目录下的内容：FileGetter.java。（进行了简单的修改，改成了对 xls 文件的读取）

iv. Controller 目录下的内容：indexController.java（由助教提供，没有进行修改）

v. Service 目录下的内容：indexService.java（进行了大幅度的改变，是本次 project 的核心内容）

2. ALGORITHM ANALYSIS

介绍本次 project 中所用到的算法并解释使用该算法的原因。

A. Dijkstra 算法，在本次 project 当中用于三个地方：最少步长、最短时间、最少换乘。

i. 最少步长：找到距离起始地点最近的起始站点，然后对起点运行 Dijkstra 算法，得出一条从起站点到始站点的最短路径。

ii. 最短时间：将起始点加入到图当中，并对起点、每个站点构建一条边，每个站点、终点构建一条边，形成一个新的图。并对起点运行 Dijkstra 算法，得到最短路径。（以上两种情况都是以时间为权重来构建的边）

iii. 最少换乘：以 15 个地铁站为顶点，构建一个图，地铁站之间相交时在两个地铁站之间构建一条边，边的权重均为 1。在执行最少换乘时，会取距离起始点最近的各三个点，共九种取点情况。结合每个起始点可能处于多条地铁线上（最多三条，最少一条），即最复杂的线情况（要从几号线走到几号线）为 81 种，最少为 9 种，为常数项。当得到了起始点所在的地铁线时，对起点所在的地铁线运行 Dijkstra 算法（借助开头所介绍的对地铁站构造的图），得到一条到达终点所在的地铁线的最短线路（必定为换乘最少，易于理解）。然后借助最短线路找出具体的最少换乘的具体路线。

B. 为什么使用 Dijkstra 算法以及实现最少换乘的思路。

i. 为什么使用 Dijkstra？显然广搜和深搜都不适用于该 project，因为本次的图是一个带权图（权重为正），广搜要求的是边的权重相等，显然本次 project 无法满足这个要求，因此广搜是不适用的。至于说 Bellman-Ford 算法，本次 project 可以使用，但是 Bellman-Ford 算法的复杂度为 O（VE），而结合优先队列的 Dijkstra 算法的复杂度可以优化到 O（（V+E）lgV）。所以使用 Dijkstra 算法是相对于 Bellman-Ford 算法更优，而且本次的所有边权重都是正值，不存在负值的情况，不需要考虑可能出现的负权重导致的死循环问题（栈溢出）。所以对于本次的单源最短路径的问题，Dijkstra 算法是最优的。

ii. 如何实现最少换乘。其实在上面有介绍最少换乘时如何实现的，那么为什么要这么做呢？首先分析一下我的算法的复杂度，取三个值 V 表示顶点，E 表示边，L 表示线路，得出最少换乘地铁线路时使用的是 Dijkstra 算法（其实可以用广搜实现，因为我设置的权重都是一样的，但是在实现时遇到了问题，因此还是借助了 Dijkstra 算法来实现），优化后的 Dijkstra 算法是 O（L^2lgL），得出了最短换乘地铁线后，我需要做的是结合最短换乘地铁线找到一条具体的最短换乘路线（前者只包含地铁线，后者包含的是具体的地铁站点），这里的时间开销主要在 getAddress 函数上（详情请看代码），我对与最短换乘地铁线只有一条的情况进行了特殊处理，从终点地铁站点进行前驱回溯（限制条件为前驱节点必须在该地铁线上），直到找到起点地铁站为止，复杂度为 O（V）（V 不会超过 40）。对于最短换乘地铁线至少两条的情况，我对它们进行了递归处理，每次取两条地铁线（1、2），从起点地铁站开始，往附近的相邻地铁站递归，并设置前驱节点（收缩条件：相邻地铁站必须在第一条地铁线上，而且相邻地铁站不能是自己的前驱节点），直到找到一个在第二条地铁线上的地铁站为止，改变起点地铁站为此站点，并取另两条地铁站（2、3）来接着递归，直到找到终点为止。则时间复杂度为 O（V/L*L）。所以总的来说，时间复杂度为 O（L^2lgL）+O（V）

iii. 如何实现最少步长和最少时间。

1. 最少步长。取距离起始点最近的起始站点，对开始站点运行 Dijkstra 算法，对于其他的每个点而言，在运行完 Dijkstra 后，里面都会存放从开始站点到这个点的最短路径（存放在 Vertex 类的 path 属性里面，path 是一个 LinkedList<Vertex>的值），然后我们循环读取 path 即可。算法时间复杂度就是 Dijkstra 的时间复杂度：O（（V+E）lgV）；
2. 最少时间。首先，根据经纬度计算出每个（共 324 个地铁站）地铁站到起始点的距离，然后根据助教所提供的速度（indexService.java 里面有一个全局变量 velocity，存放着人的速度），然后将起始点作为新顶点加入到图当中，并对每个地铁站，与起始点构建一条边，权重为距离/速度（起点到每个地铁站，每个地铁站到终点）。顺便一提，每次运行 case1 case3 的时候，我都会取一个全局变量 graph 的

拷贝 graph1 和 graph2，因此不必担心因为 Dijkstra 的执行导致图的某些性质的变化。时间复杂度也是 Dijkstra 的时间复杂度：O（（V+E）lgV）；

3. IMPLEMENTATION'S PERFORMANCE

对于本次实现的三个功能：最少换乘，最短时间，最少步长，下面将给出对于我给定的测试数据的截图（起点为复旦大学张江校区，终点为人民广场）：

步行最少：

运行时间为（第一个时间为测试整个 case 的时间，第二个为测试算法的时间）：

```
复旦大学张江校区
121.604569
31.196348
人民广场
121.478941
31.236009
1
Run time is 538423ns
The run time is 2232961ns
```

整个 case 的 run time 大概是在 2-4ms 左右（初次测试时误差偏大，多次测量后值才趋于稳定）

Dijkstra 算法的 run time 大概稳定在 0.5-0.7ms 左右（多次测量后的稳定值）

时间最短：

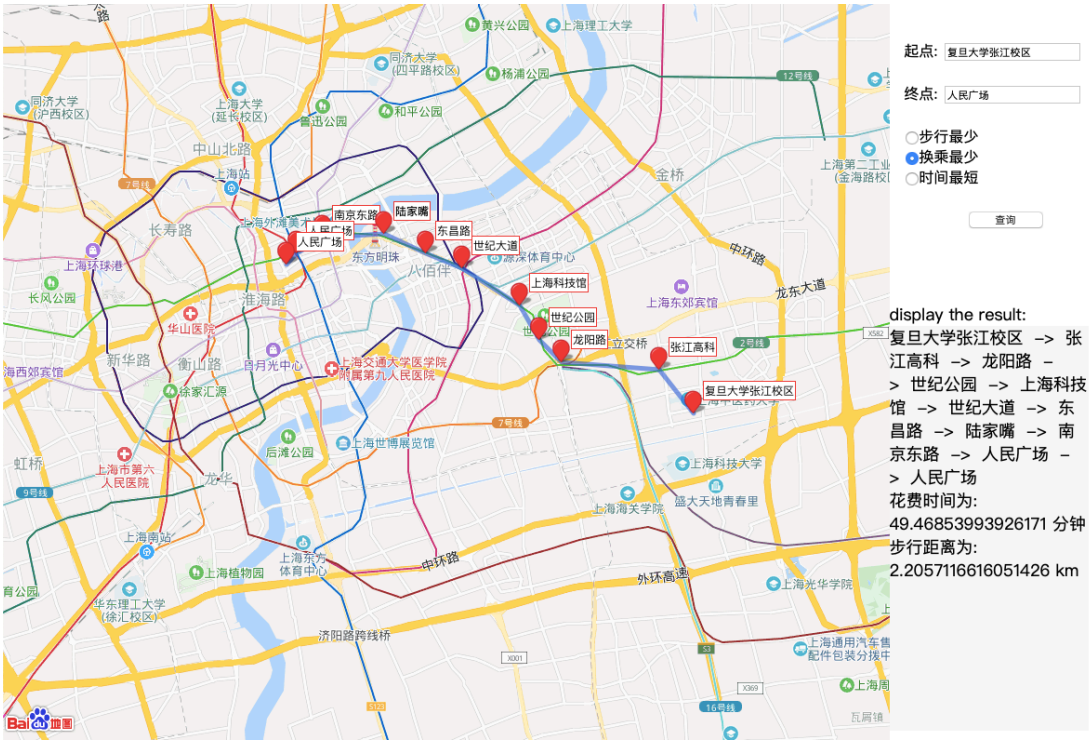
运行时间（第一个时间为测试整个 case 的时间，第二个为测试算法的时间）：

```
复旦大学张江校区
121.604569
31.196348
人民广场
121.478941
31.236009
3
Run time is 744872ns
The run time is 2585036ns
```

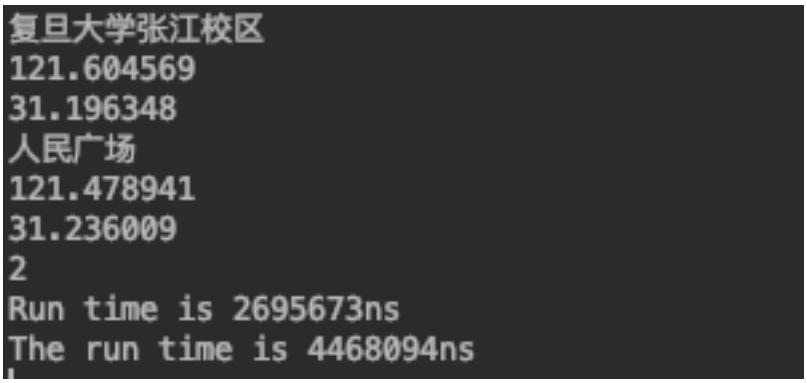
整个 case 的 run time 大概是在 2-4ms 左右（初次测试时误差偏大，多次测量后值才趋于稳定）

Dijkstra 算法的 run time 大概稳定在 0.6-0.7ms 左右（多次测量后的稳定值）

换乘最少：



运行时间（（第一个时间为测试整个 case 的时间，第二个为测试算法的时间）：



整个 case 的 run time 大概是在 4-5ms 左右（初次测试时误差偏大，多次测量后值才趋于稳定）

算法的 run time 大概稳定在 2-3ms 左右（多次测量后的稳定值）

解释一下最少换乘理论上时间快于最短时间和最小步长，但实际并非如此的原因。因为最少换乘需要对 9 组点，以及可能出现的 9 种地铁线路进行循环操作，所以总的时长要乘以一个常数 K（K<81）。所以会出现最少换乘在时间上反而超过最短时间和最少步长的情况。

4. RUN TIME TABLE

A. 步行最少（表格中的测试结果均为多次测量后取得稳定值）

起点	起点经度	起点纬度	终点	终点经度	终点纬度	Case run time（ns）	Algorithm run time（ns）
复旦大学张江校区	121.604569	31.196348	人民广场	121.478941	31.236009	2232961	538423
长宁区, S20(外环高速)	121.353674	31.240927	闵行区, 水清路, 1138 号-临	121.380982	31.136141	2598579	822870
复旦大学张江校区	121.604569	31.196348	杨浦区, 国定路, 320 弄-18 号	121.518471	31.30315	1677245	300085
杨浦区	121.503236	31.331532	浦东新区, 华夏西路辅路	121.555265	31.167292	1685630	246678
徐汇区, 虹漕路, 380 号	121.416136	31.180887	浦东新区, 灵岩南路, 1685 号	121.513009	31.141581	1484450	169700
黄浦区, 高墩街, 48 号	121.491933	31.232429	浦东新区	121.521397	31.194504	1497550	169773
杨浦区, 国权路, 242 号	121.517517	31.295398	浦东新区, 迎春路, 971 号	121.56351	31.231688	1516688	181071
杨浦区, 国和路, 1013 号	121.534016	31.323896	浦东新区, 杨思路, 885	121.506995	31.169778	2088624	330164

B. 最少换乘

起点	起点经度	起点纬度	终点	终点经度	终点纬度	Case run time（ns）	Algorithm run time（ns）
复旦大学张江校区	121.604569	31.196348	人民广场	121.478941	31.236009	3023785	1587995
长宁区, S20(外环高速)	121.353674	31.240927	闵行区, 水清路, 1138 号-临	121.380982	31.136141	3064312	1395142
复旦大学张江校区	121.604569	31.196348	杨浦区, 国定路, 320 弄-18 号	121.518471	31.30315	2225359	789040
杨浦区	121.503236	31.331532	浦东新区, 华夏西路辅路	121.555265	31.167292	2801841	969570
徐汇区, 虹漕路, 380 号	121.416136	31.180887	浦东新区, 灵岩南路, 1685 号	121.513009	31.141581	2433085	1123577
黄浦区, 高墩街, 48 号	121.491933	31.232429	浦东新区	121.521397	31.141581	1978550	574431
杨浦区, 国权路, 242 号	121.517517	31.295398	浦东新区, 迎春路, 971 号	121.56351	31.231688	1820390	470577
杨浦区, 国和路, 1013 号	121.534016	31.323896	浦东新区, 杨思路, 885	121.506995	31.169778	1861428	1785

C. 时间最短

起点	起点经度	起点纬度	终点	终点经度	终点纬度	Case run time（ns）	Algorithm run time（ns）
复旦大学张江校区	121.604569	31.196348	人民广场	121.478941	31.236009	1947668	475471
长宁区, S20(外环高速)	121.353674	31.240927	闵行区, 水清路, 1138 号-临	121.380982	31.136141	2636536	755828
复旦大学张江校区	121.604569	31.196348	杨浦区, 国定路, 320 弄-18 号	121.518471	31.30315	2734634	526157

杨浦区	121.503236	31.331532	浦东新区, 华夏西路辅路	121.555265	31.167292	2202447	384746
徐汇区, 虹漕路, 380 号	121.416136	31.180887	浦东新区, 灵岩南路, 1685 号	121.513009	31.141581	2702603	340290
黄浦区, 高墩街, 48 号	121.491933	31.232429	浦东新区	121.521397	31.141581	2256683	341272
杨浦区, 国权路, 242 号	121.517517	31.295398	浦东新区, 迎春路, 971 号	121.56351	31.231688	1793385	260191
杨浦区, 国和路, 1013 号	121.534016	31.323896	浦东新区, 杨思路, 885	121.506995	31.169778	2634911	365866

D. 分析

总的来说，各取了 8 组点之后，Dijkstra 算法的 run time 基本稳定在（因为最短步长和最少时间用的核心算法都是 Dijkstra 算法）0.1-0.7ms 左右，符合预期。

至于说最小换乘，算法的 run time 有较大的波动，原因很简单，因为 K 的取值是不一定的，K 介于 1-81 之间，因为我的代码有约束处理（而且还有一种极其特殊的情况处理），如果某种情况下换乘次数为一直接跳出循环，所以 K 的值是不定的。所以波动较大，算法的时间在 0.4-1.6ms 之间。

极端情况，我在运行最小换乘的时候，如果初始化时发现最近的三个起始点恰好取出了在一条线上的点，就直接进行查点，不会进入循环，所以才会出现 1700ns 的极端情况（我没有考虑时间问题，在运行最小换乘的时候）。