華中科技大學

课 程 实 验 报 告

课程名称：C++程序设计

实验名称：面向对象的公交地图导航

院　　系 ：计算机科学与技术

专业班级 ：　计卓 2101

学　　号 ：　U202112071

姓　　名 ：　王彬

指导教师 ：　金良海

2022 年 **12** 月 **13** 日

一、需求分析

1. 题目要求

假定所有公交车辆从起点到终点都是双向非环路的，且双向线路的所有停靠站点都对应相同。设有 M 路公交路线，第 j 路公交路线有 N_j 个站点。所有公交线路累计共有 S 个站点，第 k 个站点的坐标为（X_k，Y_k），所有坐标均以米为单位标注。邻近站点之间的距离指的是站点坐标之间的欧几里得距离。现有一人处于起点坐标（X_b，Y_b），此人需要步行到最近站点乘车，下车后要步行到达的终点坐标为（X_e，Y_e），而他特别不愿意走路，能坐公交就尽量坐公交。假定公交转乘时的步行距离为 0，试编程求他从起点（X_b，Y_b）到终点（X_e，Y_e）的乘坐线路。建立模型时需要考虑如下几种情形：（1）最少转乘；（2）最短距离。

所有公交线路的站点坐标存放于“stops.txt”文件，其中第 1 行为站点总个数，第 2 行为第 1 个站点的坐标，第 3 行为第 2 个站点的坐标，以此类推。可用图形化的界面显示站点及公交线路。“stops.txt”文件的内容如下。

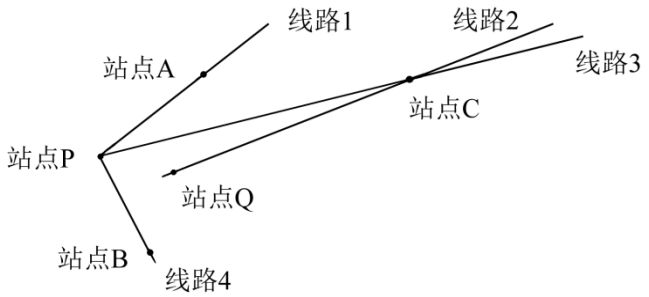
```
39
235     27
358     29
480     34
155     36
222     64
282     62
413     60
457     63
483     60
560     69
131     87
349     61
314     97
420    107
487    125
620    107
666     79
186    107
270    120
350    141
```

383 148
370 164
442 179
496 171
555 167
651 155
775 184
678 272
208 156
296 161
356 190
493 202
490 229
504 262
457 269
249 196
155 190
103 171
112 241

所有公交线路信息存放于“lines.txt”文件，其中第 1 行为公交线路总数，第 2 行为每条公交线路的站点总数，第 3 行为线路 1 经过的站点编号（对应站点坐标参见“stops.txt”），第 4 行为线路 2 经过的站点编号，以此类推。“lines.txt”文件的内容如下。

6
13 11 8 9 7 7
1 6 13 20 22 21 14 8 3 9 15 24 32
4 5 6 12 7 8 9 10 16 26 28
11 18 19 20 21 23 33 34
38 37 36 31 23 24 25 26 27
2 12 13 19 29 37 39
30 31 35 33 25 16 17

采用 Dijkstra 最短路径算法是不合适的，因为它仅考虑了距离而未考虑公交线路行驶约束。如下图所示，对于某站点 P 周围的站点 Q，其欧几里得距离虽近，但可能需要很远的转乘才能到达。例如，从站点 P 出发，要从线路 3 经站点 C 转线路 2 才能到最近的站点 Q，因为站点 P 和站点 Q 之间无直达的公交线路。



通过类型抽象形成站点、公交线路、转乘站点、转乘线路、转乘矩阵、公交系统等类，输入上述文件初始化站点和线路对象，然后通过图形化的界面显示站点和线路地图。用户用鼠标左键在地图上设定起点、鼠标右键确定终点，按照设定的最少转乘或者最短距离选项规划出从起点步行到最近站点上车、到离终点最近站点下车步行到终点的线路，在地图上用不同颜色显示规划线路若干秒，然后消除并恢复原始地图线路的颜色。

假设某个机构或单位的信息存储在“organization.txt”文件，第 1 列存放单位名称，第 2 列单位坐标。“organization.txt”文件中的格式如下，可自己添加更多单位或坐标。

华中科技大学　　990, 370
华乐山庄　　500, 340
光谷中心花园　　631, 367
光谷街北路　　766, 472

用户可输入“华科大”或“华中科大”查询其所在位置，通过最大公共子串算法进行模糊匹配，找到“华中科技大学”及其坐标。在确定始发单位坐标和终点单位坐标后，按照设定的最少转乘或者最短距离选项规划线路，并在地图上用不同颜色显示规划线路若干秒，然后消除并恢复原始地图线路的颜色。

提示：可以构造公交线路转乘矩阵 A（即 A^1）。假定有 5 条公交线路，若线路 i 和线路 j（i≠j）有 r 个转乘站点（即 r 种走法），则矩阵 A^1 的元素 $a^1[i,j]=r$；如 i=j 则规定无须转乘，即有 $a^1[i,i]=0$。则对于上述前 5 条公交线路，从公交线路 i 一次转乘到线路 j，可得如下转乘矩阵 A^1。

0　3　3　1　0
3　0　3　2　1
3　3　0　0　1
1　2　0　0　1
0　1　1　1　0

由上述转乘矩阵 A^1 可知，无法从线路 1 转乘到线路 5，因为 $a^1[1,5]=0$。可以尝试从线路 1 转乘其他线路，再从其他线路转乘线路 5，即从线路 1 经过两次转乘到线路 5。这只需要进行矩阵乘法运算 $A*A=A^2$，从线路 1 经过两次转乘到线路 5，可从 A^2 中得到共有 $a^2[1,5]$种走法。

$$a^2[1,5]=\sum_{k=1}^{5} a^1[1,k]*a^1[k,5]$$

由上述转乘公式，可计算得到 A^2，最后设置 $a^2[i,i]=0$，修正 A^2 使同一线路不用转乘。修正后的 A^2 如下。

0	11	9	6	7
11	0	10	4	5
9	10	0	10	3
6	4	10	0	2
7	5	3	2	0

由上述矩阵可知 $a^2[1,5]=7$，即从线路 1 经过两次转乘到线路 5 共有 7 种走法：（1）从线路 1 到线路 2 共有 3 个转乘点，再从线路 2 经唯一转乘点至线路 5，一共有 3 种转乘方法；（2）从线路 1 到线路 3 共有 3 个转乘点，再从线路 3 经唯一转乘点至线路 5，一共有 3 种转乘方法；（3）从线路 1 经唯一转乘点至线路 4，再从线路 4 经唯一转乘点至线路 5，一共有 1 种转乘方法。

依此类推，可以得到 A*A*A（即 A^3，反映从线路 i 经过 3 次转乘到线路 j 共有几种转乘走法）。对于本题来说，由于总共只有 M=7 条公交线路，故无重复公交的转乘最多只能转乘 M-1 次，所以只需计算到 $A^{M-1}=A^6$ 为止。任意两条线路之间 1 次、2 次……6 次转乘的走法共有 A^+种，$A^+= A^1+ A^2+ A^3+ A^4+ A^5+ A^6$。

上述 A^+运算是一种修正的矩阵“闭包”运算，可以抽象成矩阵类的一个运算。上述 A^+运算只计算了有几种转乘走法，当然，还可以同步 A^n 建立另外一个矩阵，对应元素为某种链表指针类型，用于记载对应转乘线路和转乘站点。

得到上述转乘“闭包”矩阵 A^+以后，只需要搜索离起点最近的站点（可能不止一个站点），找到最近站点所在的线路 i（可能不止一条），并搜索离终点最近的站点（可能不止一个站点），找到最近站点所在的线路 j（可能不止一条），然后在 A^+中找出 a[i, j]所描述的所有转乘线路和转乘站点即可。利用站点坐标可以得到两两站点之间的距离，并利用 A^+分别得到如下两种情形的最优转乘方案：（1）最少转乘；（2）最短距离。

2. 需求分析

(1)地图数据需要通过文件加载，需要加载的数据有地图图片文件，站点文件 stop.txt，线路文件 line.txt，机构名称 Organization.txt。

(2)当加载完地点坐标文件后，在地图上应出现小红点标识地点位置，当加载完机构名称和坐标与机构名称的对应关系后，当鼠标移至小红点上时应显示该地点挂牌单位名称，鼠标一走（并且具有一定延迟之后）单位显示自动消失。

(3)所有文件加载完毕后，可以通过左键与右键点击小红点以设置起点与终点，起点和终点可在地图窗口底部边框上显示，再次单击同一红点可取消起点或终点设置。当起点和终点都设置完毕后，可双击鼠标规划最短路径并在地图闪烁提示。

(4) 用户也可以鼠标右键弹出菜单，在出现问路窗口后，输入起点单位名称和终点单位名称，点击查询后在地图上闪烁提示最短路径。在问路对话框窗口的一个输入栏内输入单位名称时，可在输入栏下的下拉列表框中显示相似名称的单位。随着输入的汉字的增多，输

入栏下的下拉列表框中的单位逐步变少，若下拉列表框中只剩一个单位时，则默认在输入栏中选中该单位，也可以通过移动鼠标在下拉列表框中双击选中该单位。下拉列表框可随其输入栏获得焦点而出现，下一个输入栏获得焦点则隐藏前一个输入栏的下拉列表框，按钮等其它控件获得焦点则可隐藏所有输入栏的下拉列表框。

(5)进行一次查询后，在第二次查询前应擦除第一次标记的路线。

二、系统设计

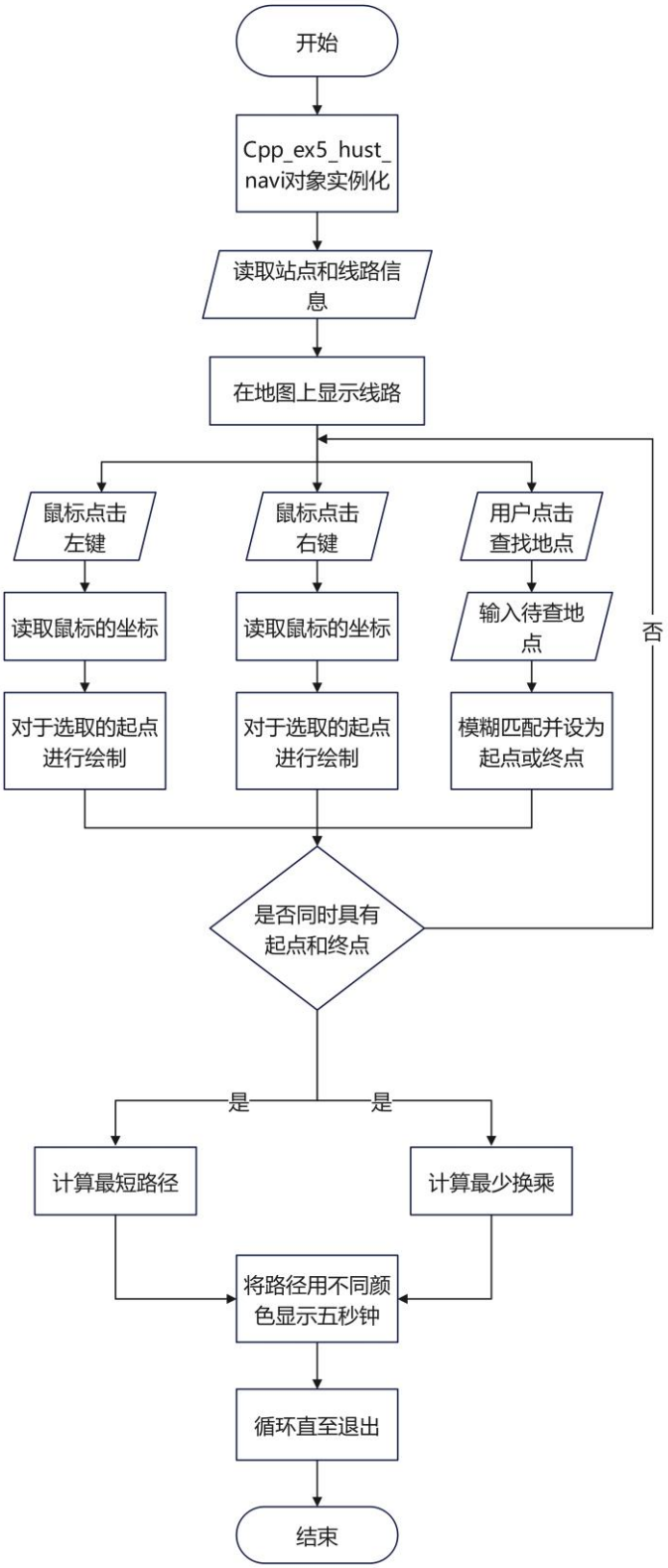
1. 概要设计

本次实验采用 Qt 并以面向对象的方法设计和编程。我们将项目分为了主窗口模块 Cpp_ex5_hust_navi、输入窗口模块 InputWidgets、逻辑层 logiclayer.h(*.cpp)、模糊查找模块 search_widget。

我们定义了 Cpp_ex5_hust_navi 作为总类。为了绘制线路图和站点图，它派生出 MyItem 类和 MyScene 类。为了计算最短距离和最少换乘，我们定义了 GIS 类作为总的计算最短路程与最少换乘的方法，根据题意我们使用矩阵相乘求和的方式进行计算。

在 Qt 中我们以 Cpp_ex5_hust_naviClass 作为主窗体，以 QMainWindow 为基类，并定义 search_widget 为查询窗体，InputWidgets 为输入窗体，SuspendForm 为悬浮在主窗体上的提示窗体。

我们将其主要流程图绘制如下。


开始
Cpp_ex5_hust_navi对象实例化
读取站点和线路信息
在地图上显示线路
鼠标点击左键
鼠标点击右键
用户点击查找地点
读取鼠标的坐标
读取鼠标的坐标
输入待查地点
对于选取的起点进行绘制
对于选取的起点进行绘制
模糊匹配并设为起点或终点
否
是否同时具有起点和终点
是
是
计算最短路径
计算最少换乘
将路径用不同颜色显示五秒钟
循环直至退出
结束

2. 详细设计

(1)Cpp_ex5_hust_navi 类的实现：

在 Qt 开发环境下定义 Cpp_ex5_hust_navi 类，以 QMainWindow 为基类来实现主界面。在 Qt Designer 中对 ui 文件进行编写和绘制，并且在菜单栏添加“文件(F)”和“查询(Q)”两个选项，分别可以使用快捷键进行快速调用。“文件(F)”下含有两个子菜单，点击则可以读入地图信息和线路站点信息； “查询(Q)”下含有三个子功能“最短距离” “最少换乘” “模糊查询”，点击则会弹出窗口分别求出该线路下的从起点到终点的最短距离、最少换乘和模糊查询结果。

在类中定义槽函数实现距离优先和最少换乘二选一，重载虚函数实现鼠标事件、键盘事件、绘图时间和计时器事件，实现流程图中功能。绘制公交信息时，线路由红色虚线表示，公交站点由红色实心点表示；绘制路线时，棕色代表起点，黄色代表终点，路径由绿色或紫色实线表示。

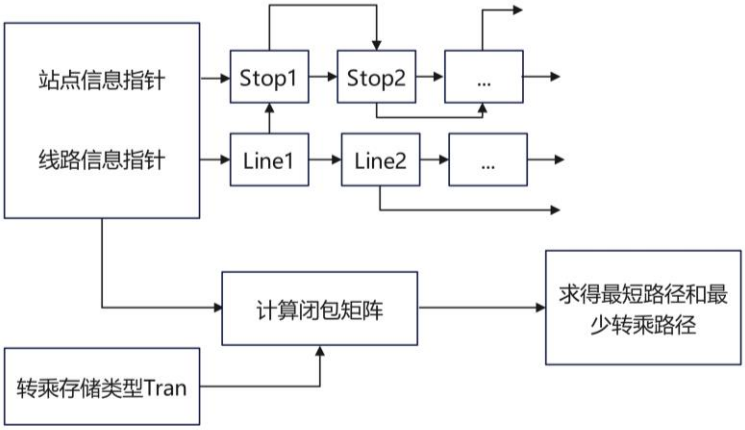
(2)MatPathCalc 类的实现：

MatPathCalc 是本项目中集成运算以求得最短路径和最少换乘的类成员。该类中保存了所有关于地图的数据和坐标数据，包括各个线路和站点的信息及其之间的关系。

MatPathCalc 类的数据成员有用以保存地图文件、坐标文件、机构名称文件、坐标与机构名称间的对应关系的文件、加载地点坐标之间连通性的文件的存储路径的变量。它有用以保存站点信息的 Stop 类型的静态指针 st 用以存储所有的公交站点，用以保存公交线路的 Line 类型的静态指针 ls，用以保存转乘闭包运算矩阵的之间对应关系的 Tmap 类型的矩阵。其中 st 直接以链表的方式保存 Stop 类型的公交站点即。我们在该类中定义了计算最短距离和最少换乘的方法 miniTran 和 miniDist 以求得其结果。

为了辅助该结构的运作，我们同时构建了一系列辅助结构。我们有用于存储站点的坐标信息和序号信息的类 Stop，有用于存储线路信息与计算线路之间的换乘次数与相交关系的 Line 类，也定义了换乘类 Tran 及换乘路径类 Route 以方便存储换乘信息。为了计算转乘的可达性矩阵和距离矩阵，我们定义了 Node 类矩阵类型和 Tmap 闭包矩阵类型。为了开发这一系列的类型，我们为每一个类都提供了深拷贝和移动赋值函数、操作符重载函数、构造函数与析构函数，用以便于本系统的开发。

MatPathCalc 类的示意图如下：

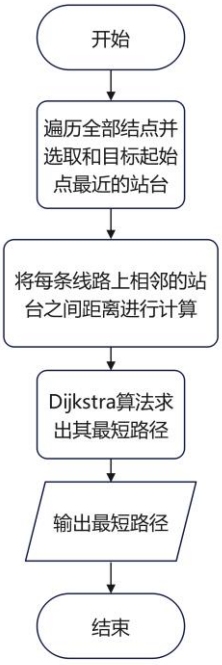


(3)最短路的求解思路：

最短路的求解有两种方式，一种是距离优先，另一种是换乘次数最少。距离优先的最短 路很容易求解，采用 Dijkstra 算法并且添加数组记录路径即可。换乘次数最少的最短路需要在 Dijkstra 算法的基础上采取一些变化。

首先将同一公交线路上的点两两之间添加一条距离为相应距离长度的边（其他正常边距离都大于 1），然后再用 Dijkstra 算法求解最短路并记录路径即可。这样在求解最短路时，由于同一条线路的点之间距离都是最小的，所以结果一定是换乘次数最小的。这样求解还会有一个问题，同一公交线路上的点两两之间人为的添加了一条最小边，所以最短路的结果会直接从一个点跳 跃到同一线路上另一个不相连的点。因此在存储最短路时，需要遍历一遍每两个“连续”的 点之间跳过的点，这样得出的结果才是正确的。

如果本次算法中文件具有有 n 个站点，m 条边（边就是所有线路的路径数之和，且不能复用），则时间复杂度为：O((n+m)logn+n)等。



(4)最少换乘的求解思路：

为求得换乘次数最少的最短路，我们考虑在 Dijkstra 算法的基础上采取一些变化。我们在(3)的基础上，添加一个 rou 数组以记录转乘次数，其它照旧。只需用 rou 数组替代原方案中的计算距离的方案，即可获得最少换乘。

(5)MyScene 类以捕捉鼠标和键盘事件和线路绘制：

MyScene 中数据成员有记录上次设置的起点与终点的 QPoint 类型的变量 depart 与 arrive，两者如果尚未输入则标记为 false（即未输入的状态）。

在 MyScene 中需要实现的方法有将地图和坐标点显示在屏幕上，左键释放时调用设置起点并显示在地图上，右键点击设置终点并显示，鼠标停留在站点上输出该站点相应的编号。同时本方法实现依据线路信息画线，画圆。具体实现思路如下。

画圆与画线都是利用 QBrush 中的画笔与画刷完成。注意在画点时，我们为了体现起点终点和站台点的区别，设置了画黄点、棕点与红点；红点为默认画点，画棕色点需要额外设置。

我们首先利用 css 样式文件在画框中导入对应的地图图片，并将地图图片文件加载入资源文件中以备调用。

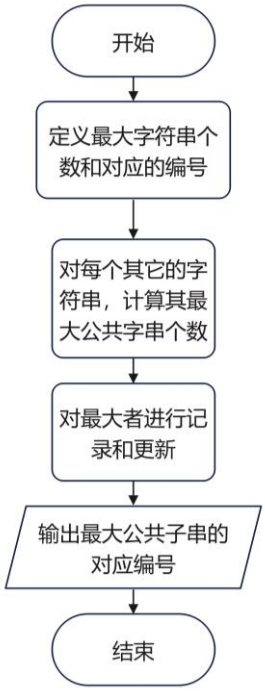
同时我们定义了 MyItem 类来实现对每一个结点站台的支持。该结点站台具有 mousePressEvent 来检测当前的图元是否已被选中，并且调用基类 QGraphicRectItem 的鼠标选中方法。在 MyItem 的构造函数中，我们说明了画笔的笔触和画笔的颜色 qpens。

为使鼠标放在地点坐标上可以显示地点信息，需要在 MyScene 中的 mouseMoveEvent () 方法进行操作。如果未加载地图文件，那么只需相应基类中的鼠标移动方法即可。我们获取鼠标的移动坐标后，对所有的结点进行查询看是否其距离和鼠标之间的少于一定数值（本实验为 8 个像素点）然后调用 SuspendForm 悬浮窗体 dlg 来把站点信息显示在站点旁边 2000ms。

为实现左键点击设置起点和右键点击设置终点，应添加鼠标点击事件响应函数 mousePressEvent，在函数中首先判断若图层是否已加载，若未加载则不响应。然后先将左键点击的点作为参数传入在 MyScene 中 AddItem 方法，将所定位的点的坐标写入 item 中并实现对起点终点的更换设置。同时，将起点或终点的标记设为 true。然后比较所有的起点 listItem，若两者不相等说明重新设置了起点，于是将该元素对应的点删去，将 depart 对应的点涂成棕色色，然后将选取的结点加入 MyItem；选取终点的过程相仿。

(6)search_widgets 类：

该类用以进行模糊匹配。我们定义了类 search_widget，其基类为 QWidget，它具有两个共有方法分别返回查询点的 x 坐标和 y 坐标。同时我们写入字符串模糊匹配方法 fuzzy_match（std::string target），同时为了传递与主模块之间的通信，定义了两个槽函数 set_start()和 set_end()和一个信号函数 send_search_pos(int, int, int kind)。其匹配过程使用 LCS 算法，大致流程图如下。



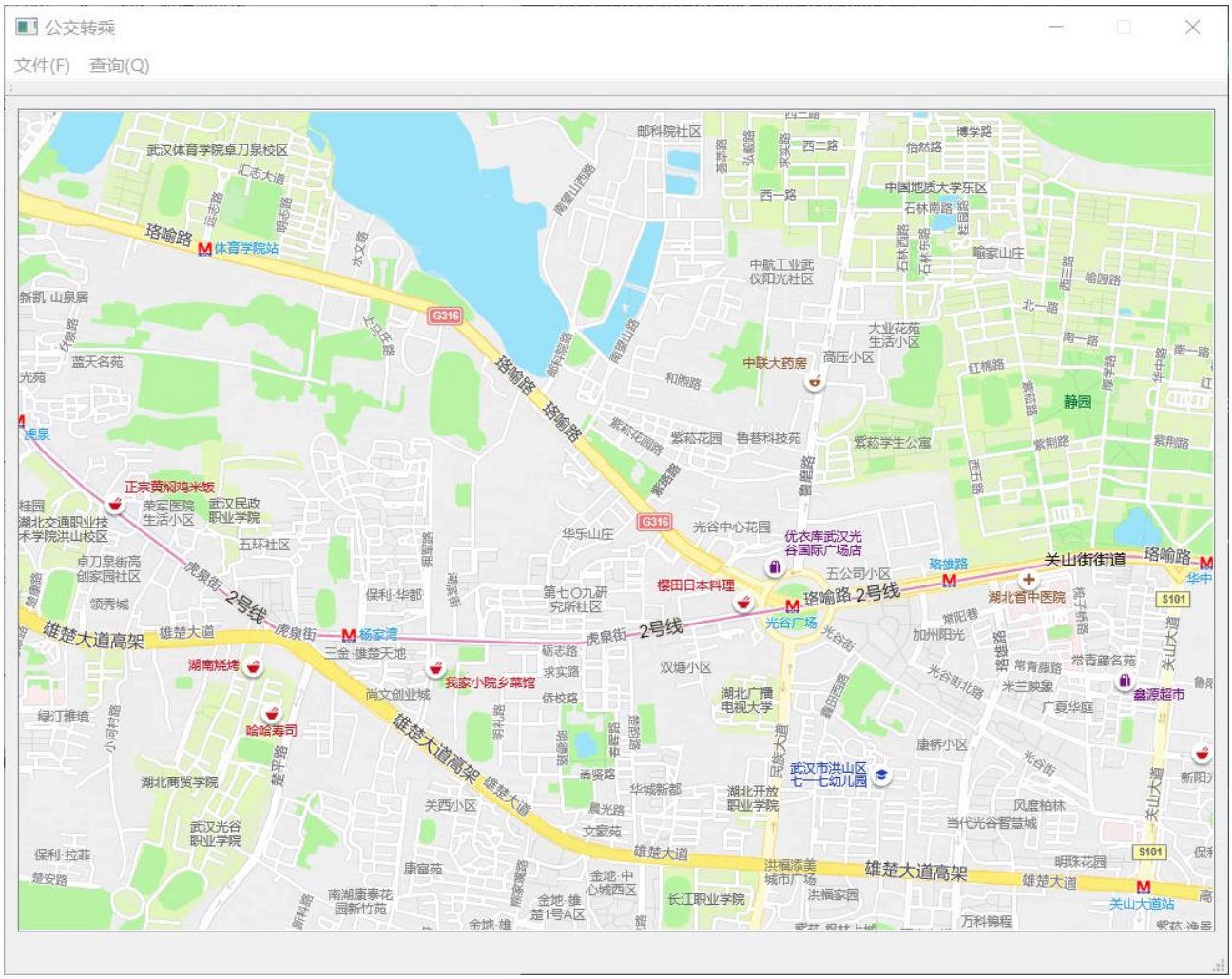
三、软件开发

实验采用 Visual Studio 2022 以及 Qt Creator 进行代码编写、编译和调试。在 Visual Studio 2022 中对 Cpp_ex5_hust_navi 类及其派生类、GIS 类进行编写。在 Qt Creator 中对 Cpp_ex5_hust_naviClass 类和 InputWidgets 类进行编写。最后由 Qt Creator 生成可执行文件。

四、软件测试

一、 主页面

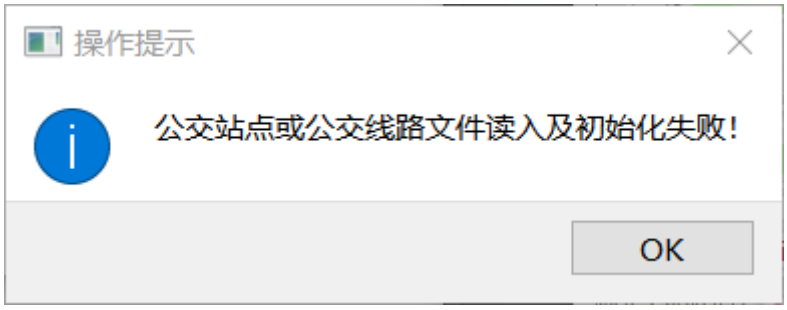
测试程序主界面运行是否正常。黑色是实心点表示公交站点，绿色虚线表示公交线路。



二、 加载文件

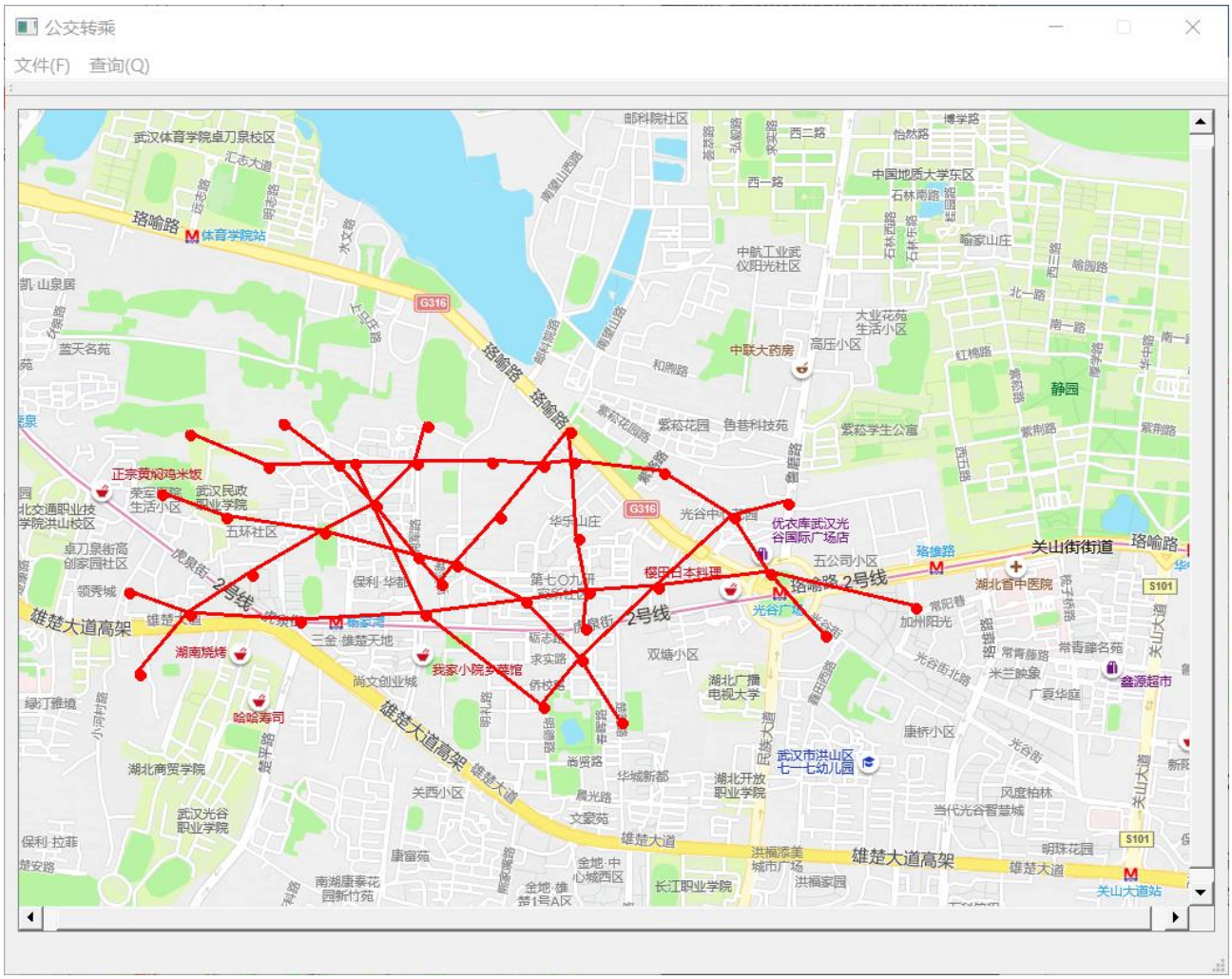
测试加载文件和路径查找是否正确。我们依次输入错误的地址和正确的地址，并查验程序是否能够对错误地址进行报错和返回。

我们在线路文件内输入了错误的地址。



程序正常报错。

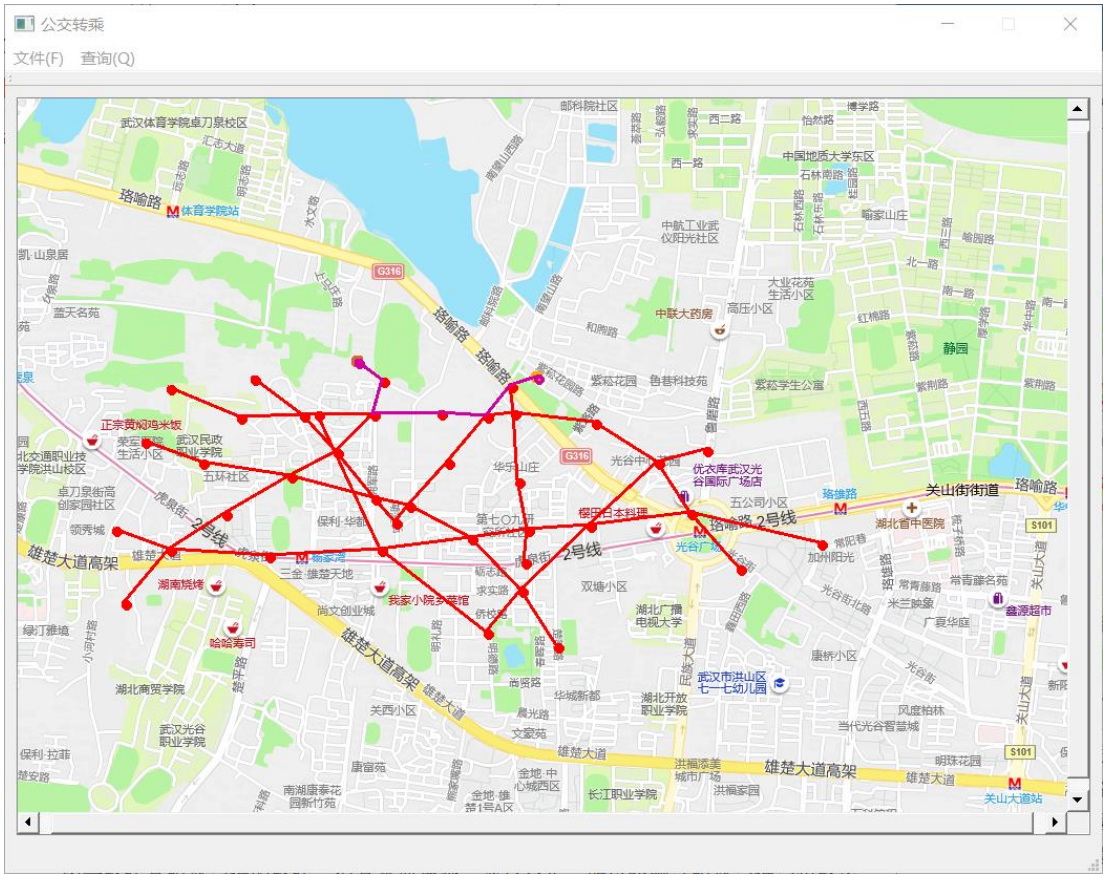
我们对程序输入了正确的地址。



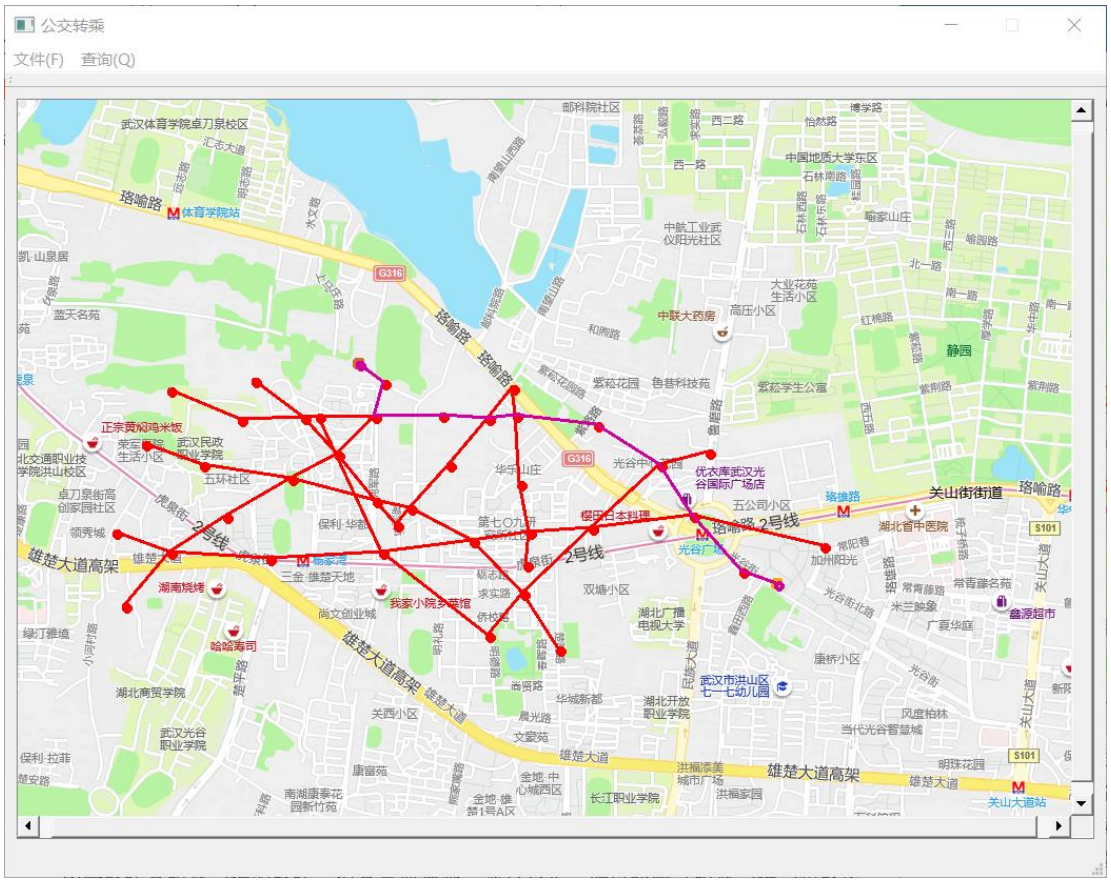
程序对路径进行了加载。可以看到公交站点和线路显示正常。

三、 最短路径绘制

测试距离最短最短路求解和显示是否正确。单击左键确定起点，单击右键确定终点。紫色实线表示最短路线，棕色实心点表示起点，黄色实心点表示终点。可以看到我们的程序输出了最短路径。

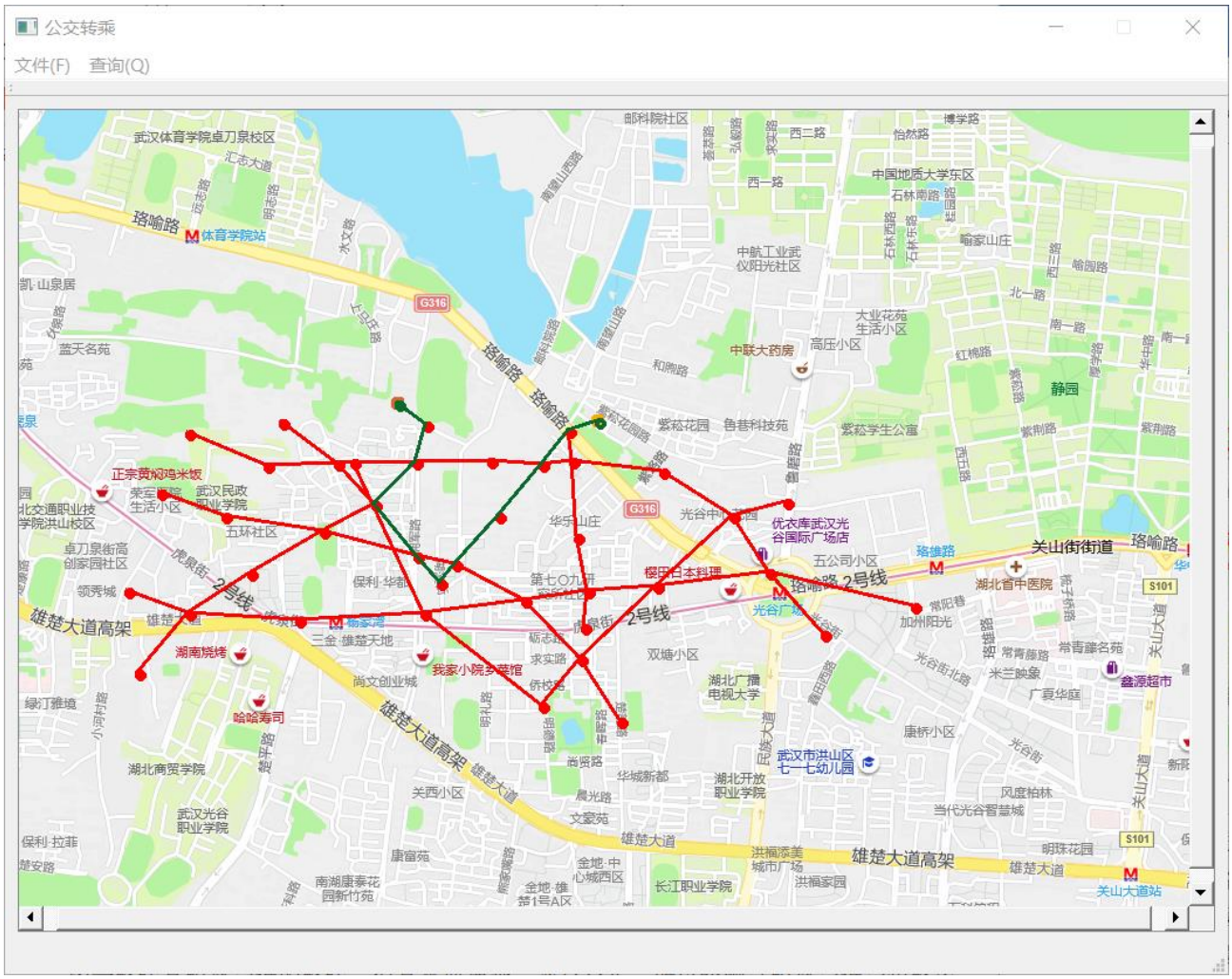


我们另外选取位于右下角的终点，进行另一组实验。可以看出程序仍然求得了最短路的解。

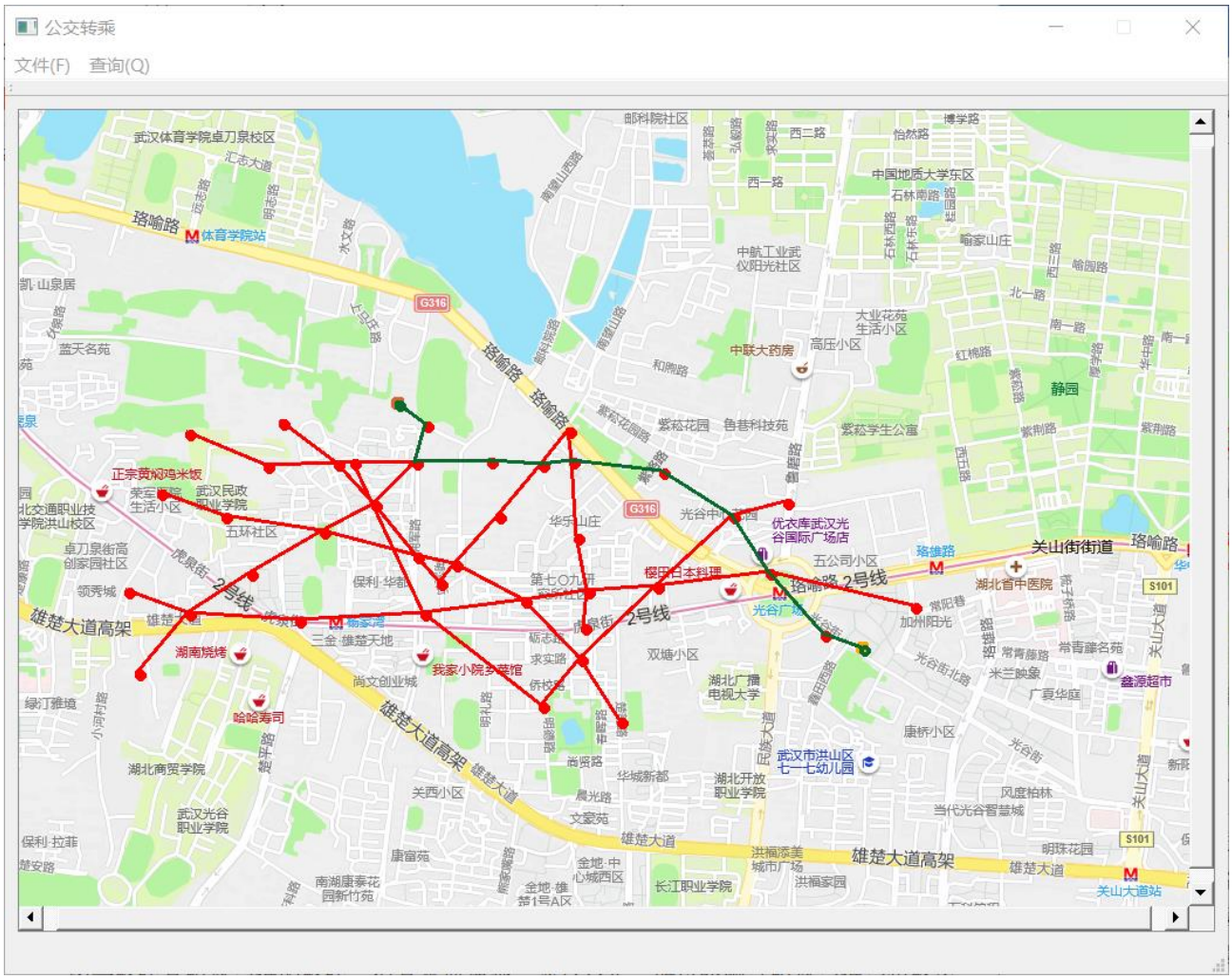


四、 最少换乘绘制

测试换乘次数最少最短路求解和显示是否正确。单击左键确定起点，单击右键确定终点。绿色实线表示最少换乘路线，棕色实心点表示起点，黄色实心点表示终点。可以看到我们的程序输出了最少换乘的路径。



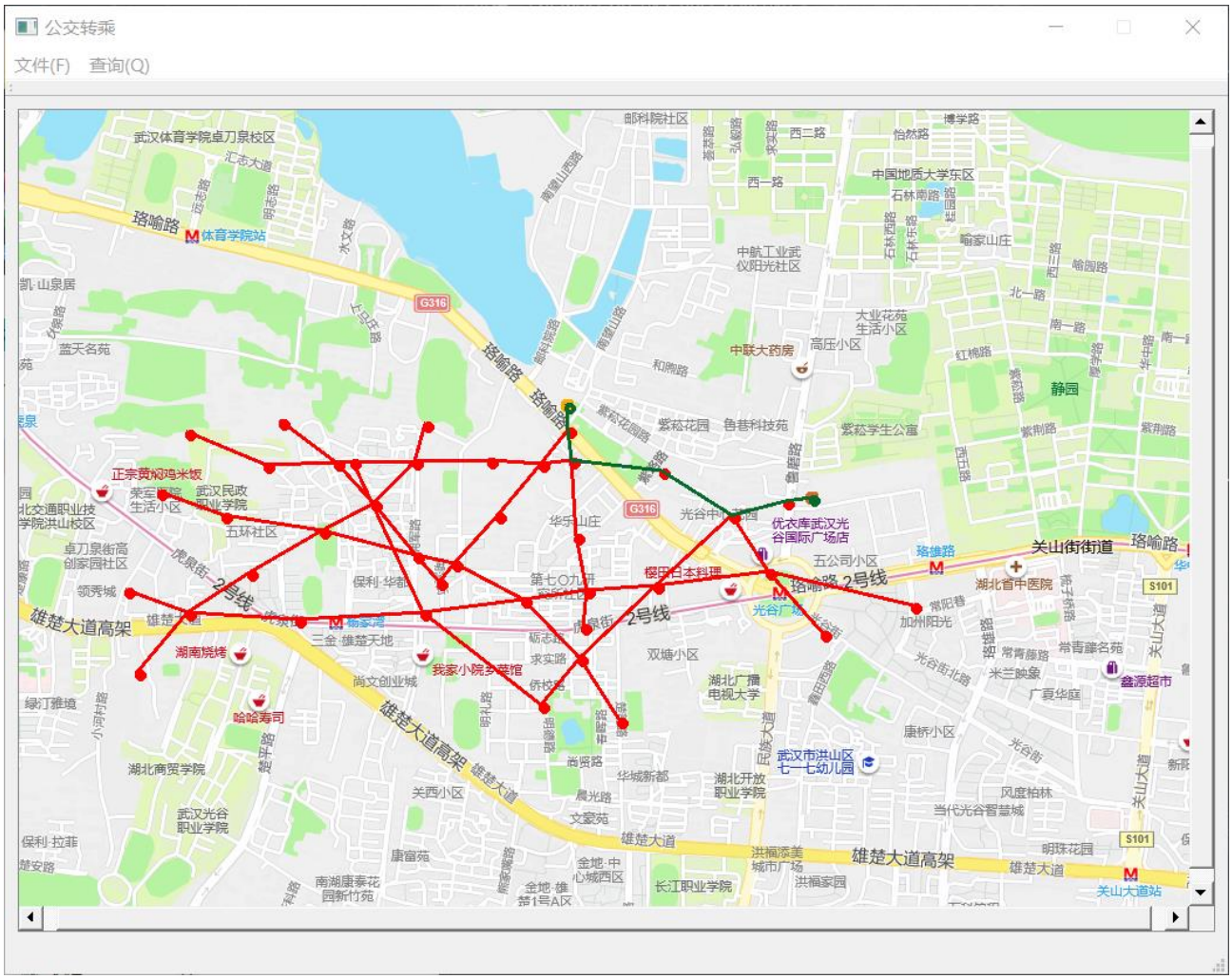
我们对另一组起点和终点进行测试。



可以看出路线并不是最短路径，可是分别只需要换乘 2 次、1 次，其换乘次数最少。由此验证了程序的正确性。

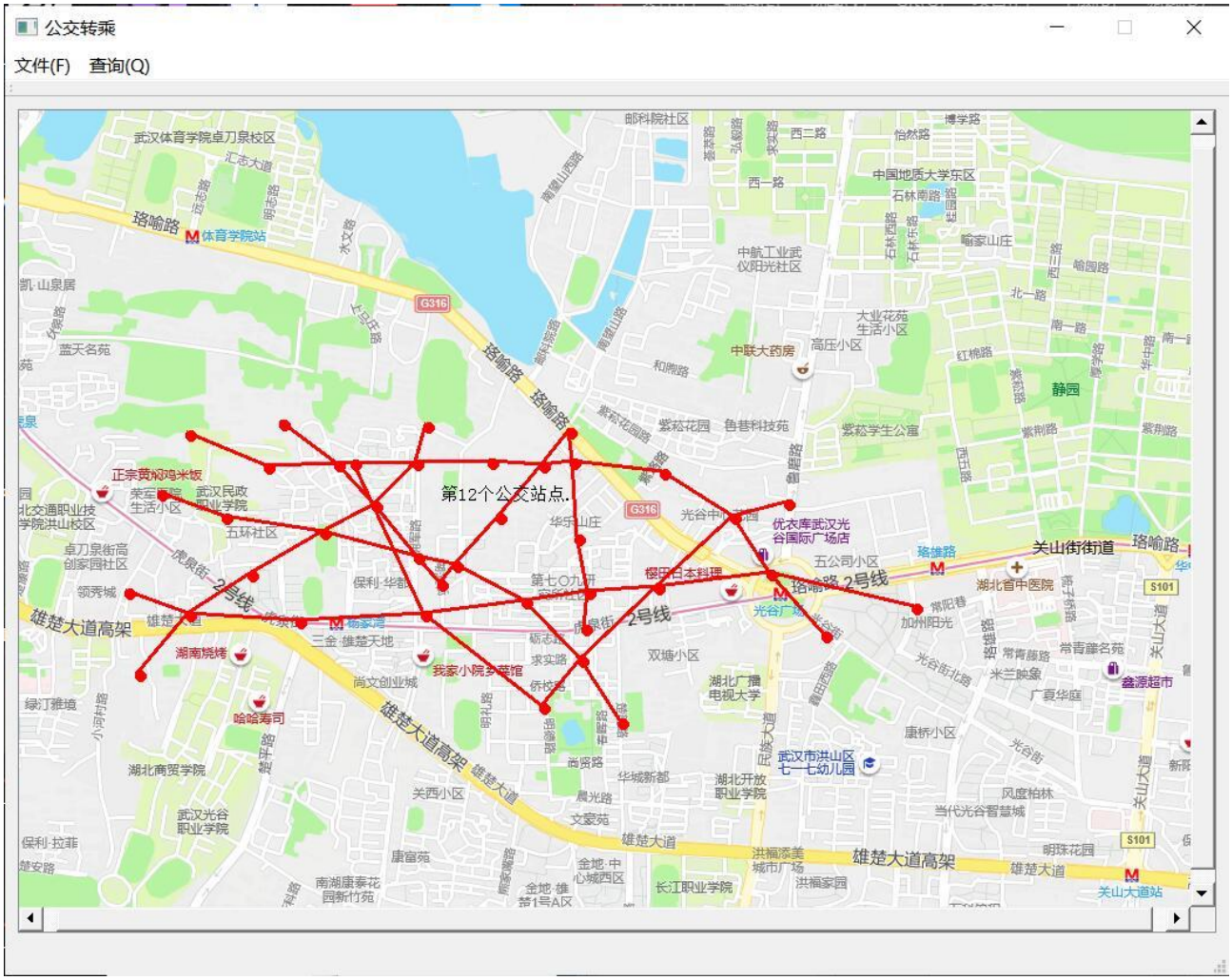
五、 自动显示测试

单击“查询”下的“查找站点”。 在文本框中输入地点缩写点击确定即可查询到地点，并将模糊匹配结果设为起点或终点。本实验由于 emit 信号至主窗体不成功，导致难以设为起点。后续可以通过重新调整主窗体和弹出窗体的继承关系进行调整。



六、 浮标测试

当鼠标悬浮在站点上方时，会显示站点的编号。



五、特点与不足

1. 技术特点

使用 Qt 进行开发，并且我们为逻辑层的站点类和线路类矩阵类进行了完备的构造、赋值的成员函数定义和运算符的重载，不易抛出错误。

2. 不足和改进的建议

我们使用的是静态地图，使得点无法对准地图上的位置，不容许用户进行修改。同时信号函数反馈至主窗体存在一些问题。

六、过程和体会

1. 遇到的主要问题和解决方法

这次设计中遇到的首先的问题是 Qt 的安装和版本不兼容，在添加 Qt 编译器时始终无法加入 Visual Studio 2022 community 版本之中。于是后来我在网上下载了其它版本的 Qt 以运行程序，终于找到了一个可以正常运行的版本。

第二个问题是在 QGraphics 内无法加载图片。如果只在 ui 文件中添加的话，实际运行时

仍然未见图片。后来我发现是 Qt 和 vs 的链接器出现了问题，又在网上寻找解决方案，调整了编译的环境变量（尤其是编译器的使用先后顺序），后来解决了这个问题。

2. 课程设计的体会

本次实验让我真正接触到了面向对象程序设计方法来设计程序，体会到了一个复杂程序包含那些类，这些类的对象之间的关系，数据成员和方法应该放在哪些类里，在完成实验的过程中我也学会了通过查阅文档来解决遇到的问题，这让我收获很多

七、源码和说明

1. 文件清单及其功能说明

在“5.rar”中，我们留存了代码文件至文件夹“code\”中，项目文件至“Cpp_ex5_hust_navi\”中，Release 版本文件“Release\”中。我们的程序运行截图放在了压缩文件包的“image\”中。

2. 用户使用说明书

点击 Cpp_ex5_hust_navi.exe，点击左上角的文件中的读取文件，点击线路位置信息时选中或浏览至 lines.txt，点击站点位置信息时选中 stops.txt。全部加载完毕后可以开始查询路径，可以在地图上左右键点击分别设置起点终点。选取完起点终点之后，在“查询”栏选中最短路径，最短路径以紫色路线标出；若选择最少换乘，最少换乘将会以墨绿色标注出。可以进行模糊匹配并设为起点或终点，其它的操作方式照旧。查询完毕后点击文件中的退出即可关闭窗体程序，或者之间点击关闭程序符以将本路径系统关闭。

3. 源代码

我们将源代码附录于下。在每一个文件的开头，我们都注释了代码所对应的文件。同时，我们只附录上了头文件(*.h)和代码实现文件(*.cpp)，Qt 自动生成的 ui 文件因实验报告的长度我们未编入报告，详细可见于报告附录的压缩文件“5.rar”。

```
/* in File main.cpp */

#pragma once

#include "Cpp_ex5_hust_navi.h"
#include <QtWidgets/QApplication>
#include <qdesktopwidget.h>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Cpp_ex5_hust_navi w;
    w.show();
    return a.exec();
```

```
}


/* in File Cpp_ex5_hust_navi.h */

#pragma once

#include <QtWidgets/QMainWindow>
#include <QGraphicsRectItem>
#include "ui_Cpp_ex5_hust_navi.h"
#include "search_widget.h"

class InputWidgets;
class Cpp_ex5_hust_navi : public QMainWindow
{
    Q_OBJECT
public:
    Cpp_ex5_hust_navi(QWidget *parent = Q_NULLPTR);
    ~Cpp_ex5_hust_navi( );
private:
    Ui::Cpp_ex5_hust_naviClass ui;
    // 站点模糊搜索模块
    search_widget* search_ui;
    InputWidgets *fl;
    QTimer* m_Timer;
    QGraphicsItemGroup* gItem;
    void deleteItems();
protected:
    void closeEvent(QCloseEvent* event);
private  slots:
    void loadmap();
    void closewnd();
    void zszc();
    void zdjl();
    void search_stop();
    // 接受站点查询的位置信息,kind=0 为起点，1 为终点
};
class MyScene : public QGraphicsScene
{

public:
    explicit MyScene(QObject* parent = 0);
    void stopLines(QGraphicsView*);
    void emitSignalToMainWindow(int xx, int yy, int od);
protected:
```

```
    QGraphicsView* qgv;
    void mouseMoveEvent(QGraphicsSceneMouseEvent* event);//覆盖 mousePressEvent 以捕获鼠标事件
    void mousePressEvent(QGraphicsSceneMouseEvent* event);//覆盖 mousePressEvent 以捕获鼠标事件
signals:
public slots:
};
class MyItem: public QGraphicsRectItem {
    int cx, cy;  //点击时的坐标
    int cf;      //左键点击=1，表示地点，右键点击=2 表示终点
    int cs;      //靠近该点的坐标个数
    int bs[6];   //最多存放 6 个站点的站点编号
public:
    MyItem(int x,int y, int f);
    MyItem& operator<<(int s);
    int operator()(int x, int y);
    int& x();
    int& y();
    int& f();
    int& c();
    int& operator[](int);
    int checkAllStops();
    void mousePressEvent(QGraphicsSceneMouseEvent* event);
};


/* in File InputWidgets.h */

#pragma once

#include <QWidget>
#include "ui_InputWidgets.h"

class InputWidgets : public QWidget
{
    Q_OBJECT

public:
    InputWidgets(QWidget *parent = Q_NULLPTR);
    ~InputWidgets();
    QGraphicsView* parnt;
    void myShow(QGraphicsView* p);
private:
    Ui::InputWidgets ui;
```

```
private slots:
    void inputStop();
    void inputLine();
    void checkFile();
};


/* in File search_widget.h */

#pragma once

#include <QWidget>
#include "ui_search_widget.h"
#include <string>
#include <vector>
#include <fstream>
#include <iostream>

typedef struct store_points sp;
struct store_points {
    std::string name;
    int x;
    int y;
};

class search_widget : public QWidget
{
    Q_OBJECT

public:
    search_widget(QWidget *parent = nullptr);

    // 没有指针成员，默认析构
    ~search_widget() = default;

    // 返回查询点的位置
    int get_x();
    int get_y();

private:
    Ui::search_widgetClass ui;

    // 站点数据库
    std::vector<sp> data;
```

```
    // 查询点的位置
    int pos_x, pos_y;

private:
    // 字符串模糊匹配,返回数据库中最大模糊匹配的下标
    int fuzzy_match(std::string target);

    // 调用模糊匹配并保存站点位置
    void search();

private slots:
    // 负责和主模块的通信
    // 传递起点
    void set_start();

    // 负责和主模块的通信
    // 传递终点
    void set_end();

signals:
    void send_search_pos(int x, int y, int kind);
};


/* in File SuspendForm.h */

#pragma once
#include<QTimer>
#include <QDialog>
#include "ui_SuspendForm.h"

class SuspendForm : public QDialog
{
    Q_OBJECT

public:
    SuspendForm(const QString& msg, QWidget *parent = Q_NULLPTR);
    ~SuspendForm();
    void startTimer(int);
private:
    Ui::SuspendForm ui;
    QTimer* m_pTimer;
    void initFrame(const QString& msg);
};
```

```
/* in File Cpp_ex5_hust_navi.cpp */

#include "Cpp_ex5_hust_navi.h"
#include "logiclayer.h"
#include "InputWidgets.h"
#include <QDialog>
#include <QDesktopWidget>
#include <QApplication>
#include <QMessageBox>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsEllipseItem>
#include "SuspendForm.h"
#include "search_widget.h"

MatPathCalc* gis = nullptr;
//地图信息
bool route=false;
//未显示查询路径时
bool depart = false;
//未选取步行起点
bool arrive = false;
//未选取步行终点

MyItem::MyItem(int x, int y, int f): QGraphicsRectItem(x-3,y-3,7,7), cx(x), cy(y), cf(f),
cs(0) {
    //以下根据鼠标左键点击步行起点和鼠标右键点击步行终点设置不同画笔颜色
    QBrush qbrush(f==1? QColor(210, 105, 45) : QColor(255, 170, 00));
    //根据 f 设置颜色
    QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin);
    //图元点的画笔
    QGraphicsRectItem* item = this;
    item->setPen(qpens);
    item->setBrush(qbrush);
    //item->setFlag(QGraphicsItem::ItemIsMovable); //不选中：因为没有移动图元点的操作
    checkAllStops();
    //检测所有站点，找出最近的站点存于 bs 中
}

int& MyItem::x() { return cx; }
int& MyItem::y() { return cy; }
int& MyItem::f() { return cf; }
int& MyItem::c() { return cs; }

int& MyItem::operator[](int x) {
```

```
    if (x < 0 || x >= cs) throw "subscription overflow for stops around departure point!";
    return bs[x];
    //返回已检测的编号为 x 的站点编号
}

int MyItem::operator()(int x, int y) {
    //仅用于检测两点距离远近，不开方
    return (x - cx) * (x - cx) + (y - cy) * (y - cy);
}

MyItem& MyItem::operator<<(int s) {
    if (s < 0 || s >= MatPathCalc::ns) return *this;
    int d = (*this)(MatPathCalc::st[s].X(), MatPathCalc::st[s].Y()); //d 为当前图元点
到站点 s 的距离
    int m;
    //m 为当前图元点到先前已检测站点的距离
    if(cs==0||d<(m = (*this)(MatPathCalc::st[bs[0]].X(),
MatPathCalc::st[bs[0]].Y()))){ //若 bs 没有元素即 cs==0，或距离站点 s 更近
        bs[0] = s;
        cs = 1;
        return *this;
    }
    if (d == m) {
        //和已检测站点比，到标号为 s 的站点的距离相同时
        if (cs == 6) return *this;
        bs[cs] = s;
        //只保存和最近距离相同的站点
        cs++;
        //距离相同的站点个数增加
    }
    return *this;
}

int MyItem::checkAllStops(){//检测所有站点，找出最近的站点存于 bs 中
    for(int c=0;c<MatPathCalc::ns; c++)
        operator<<(MatPathCalc::st[c].N());
    return cs;
    //返回距离最近且相同的站点个数
}

void MyItem::mousePressEvent(QGraphicsSceneMouseEvent* event) {
    setSelected(true);
    //当前图元被选中
    QGraphicsRectItem::mousePressEvent(event);
}
```

```
//定义自己的场景 MyScene，以便能够捕获鼠标或键盘事件
MyScene::MyScene(QObject* parent): QGraphicsScene(parent)
{
    clearFocus();
    qgv = Q_NULLPTR;
    //没有加载地图文件时
}

void MyScene::mouseMoveEvent(QGraphicsSceneMouseEvent* event) {
    //注意在其.ui 界面文件中，mouseTracking 必须勾选，否则不会出现此事件
    if (qgv == Q_NULLPTR) {
        //如果没有加载地图文件
        QGraphicsScene::mouseMoveEvent(event);
        return;
    }
    QPointF qpointf = event->scenePos();
    //获取鼠标移动时的坐标
    for (int n = 0; n < gis->ns; n++) {
        if (fabs(gis->st[n].X() - qpointf.x()) < 8 && fabs(gis->st[n].Y() - qpointf.y())
< 8) {
            //以下提示信息必须使用 fromLocal8Bit()，否则中文提示会显示乱码
            SuspendForm dlg(QString::fromLocal8Bit("第")+QString::number(n+1, 10,
0)+QString::fromLocal8Bit("个公交站点."));
            //dlg.setAttribute(Qt::WA_ShowModal,true); 若调用 show()则需设置无边框,若调
用 dlg.exec()则不用此行，
            dlg.startTimer(2000);              //设置悬停显示时间为 2 秒，时间到自动关闭
dlg
            QPointF m1 = qgv->mapToGlobal(QPoint(qpointf.x(), qpointf.y()));
            dlg.move(QPoint(m1.x(), m1.y()));
            dlg.exec();                        //显示站点提示信息
        }
    }
    QGraphicsScene::mouseMoveEvent(event);      //回调基类鼠标事件
}

void MyScene::emitSignalToMainWindow(int xx, int yy, int od) {
    if (od==1) depart = true;
    if (od==2) arrive = true;
    QList<QGraphicsItem*> listItem = items();
    for (int i = listItem.length() - 1; i >= 0; i--) {
        MyItem* item = (MyItem*)listItem[i];
        if (item->f() == od) {
            listItem.removeAt(i);
            delete item;
```

```
            break;
        }
    }
    addItem(new MyItem(xx, yy, od));
}

void MyScene::mousePressEvent(QGraphicsSceneMouseEvent* event)
{
    if (qgv == Q_NULLPTR||  route==true) {      //未加载地图及显示查询结果时，不响应鼠标按下事件
        QGraphicsScene::mouseMoveEvent(event);
        return;
    }
    QPointF qpointf = event->scenePos();    //获取鼠标坐标
    QList<QGraphicsItem*> listItem = items();
    int lb = 0;
    if (event->button() == Qt::LeftButton) { lb = 1; depart = true; }   //检查左键是否按下
    if (event->button() == Qt::RightButton) { lb = 2; arrive = true; }  //检查右键是否按下
    for (int i = listItem.length() - 1; i >= 0; i--) {
        MyItem* item = (MyItem*)listItem[i];
        if (item->f() == lb) {
            listItem.removeAt(i);
            delete item;
            break;
        }
    }
    addItem(new MyItem(qpointf.x(), qpointf.y(), lb));
    QGraphicsScene::mousePressEvent(event);         //回调基类鼠标事件
}

void MyScene::stopLines(QGraphicsView* parnt) {     //加载地图站点和线路
        //按视图 graphicsview 大小设置 scene 显示区域大大小
        QSize viewsize = parnt->size();             //取得 graphicsview 视图区域大小
        MyScene* scene;
        if (parnt->scene() != Q_NULLPTR)   delete parnt->scene();
        scene = new MyScene(parnt);                //创建 scene
        scene->setSceneRect(0, 0, viewsize.width(), viewsize.height());
        scene->qgv = parnt;
        //显示所有公交站点
        QBrush qbrush(QColor(255, 0, 0));          //设置颜色
        QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //站点的笔
        for (int n = 0; n < gis->ns; n++){
            scene->addEllipse(gis->st[n].X(), gis->st[n].Y(), 6, 6, qpens, qbrush);
```

```
        }
        //显示所有线路
        QPen qpenl(qbrush, 3, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //线路的笔
        for (int n = 0; n < gis->nl; n++) {
            LINE& line = gis->ls[n];
            int stops = line.NOFS();
            for (int m = 1; m <stops; m++) {
                STOP s = gis->st[line[m - 1]];
                STOP t = gis->st[line[m]];
                QLineF ql = QLineF(s.X(), s.Y(), t.X(), t.Y());
                scene->addLine(ql, qpenl);
            }
        }
        parnt->setScene(scene);
        parnt->show();
}


Cpp_ex5_hust_navi::Cpp_ex5_hust_navi(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);

    // 初始化子模块
    fl = Q_NULLPTR;
    gItem= Q_NULLPTR;
    m_Timer = new QTimer(this);
    m_Timer->setSingleShot(true);
    search_ui = new search_widget();//定时器只执行一次

    // 画图模块和读取地图之间的槽函数链接
    connect(ui.action_open, SIGNAL(triggered(bool)), this, SLOT(loadmap()));
    connect(ui.action_exit, SIGNAL(triggered(bool)), this, SLOT(closewnd()));
    connect(ui.action_zszc, SIGNAL(triggered(bool)), this, SLOT(zszc()));
    connect(ui.action_zdjl, SIGNAL(triggered(bool)), this, SLOT(zdjl()));
    connect(ui.action_search, SIGNAL(triggered(bool)), this, SLOT(search_stop()));
    // 中文查找模块和当前模块的通信，用于传递坐标信息
    //connect(this->search_ui, &search_widget::send_search_pos, this,
&Cpp_ex5_hust_navi::recv_search_pos);
    //以下 Lambda 表达式可以用自动类型推导代替,或用非成员函数的地址代替
    connect(m_Timer, &QTimer::timeout, this, [=]() {
        QList<QGraphicsItem*> listItem = ui.graphicsView->scene()->items();
        deleteItems();
        //查询结果显示时间一到，就删除场景的所有图元
        route = false;
```

```
        //查询结果显示完毕，可以重新选取步行起点或终点
    });
}
Cpp_ex5_hust_navi::~Cpp_ex5_hust_navi(){
    if (fl != Q_NULLPTR) {
        fl->hide();
        delete fl;
        fl = Q_NULLPTR;
        delete m_Timer;
        deleteItems();
        delete  gis;
        delete search_ui;
    }
}
void Cpp_ex5_hust_navi::closewnd() {
    if (ui.action_exit->isChecked()==false) return;
    //鼠标点击一次触发两次，第二次触发直接返回
    ui.action_exit->setChecked(false);
    //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
    if (fl != Q_NULLPTR) {
        fl->hide();
        delete fl;
        fl = Q_NULLPTR;
    }
    close();
}
void Cpp_ex5_hust_navi::closeEvent(QCloseEvent* event)
{
    if (fl != Q_NULLPTR) {
        fl->hide();
        delete fl;
        fl = Q_NULLPTR;
    }
}
void Cpp_ex5_hust_navi::loadmap() {

    if (ui.action_open->isChecked() == false) return;
    //鼠标点击触发两次，第二次触发直接返回
    ui.action_open->setChecked(false);
    //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
    if (fl != Q_NULLPTR) {
        //如果先前打开过站点及线路输入窗口
        fl->show();
        //则直接显示该窗口
        return;
```

```
    }
    arrive=depart = false;  //此时未选取步行起点或终点
    fl = new InputWidgets(); //如果以前没有打开过站点及线路输入窗口
    fl->setWindowFlags(Qt::WindowStaysOnTopHint); //设置最外层显示
    fl->myShow(ui.graphicsView);
}
void Cpp_ex5_hust_navi::deleteItems() {
    //删除场景的所有图元
    if (gItem == Q_NULLPTR) return;
    ui.graphicsView->scene()->removeItem(gItem);
    for (int i = gItem->childItems().size() - 1; i >= 0; i--) {
        QGraphicsItem* item = (gItem->childItems())[i];
        gItem->removeFromGroup(item);
        delete item;
    }
    delete gItem;
    gItem = Q_NULLPTR;
}

void Cpp_ex5_hust_navi::zszc() {
    //先计算最少转乘的路径,先获得起点坐标和终点坐标
    if (ui.action_zszc->isChecked() == false) return;
    //鼠标点击一次触发两次，第二次触发直接返回
    ui.action_zszc->setChecked(false);
    //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
    QList<QGraphicsItem*> listItem;
    if ((depart&&arrive)==false) return;
    //若没有选中步行起点和终点，则返回
    listItem = ui.graphicsView->scene()->items();
    MyItem* itemDepart = (MyItem*)listItem[0];
    MyItem* itemArrive = (MyItem*)listItem[1];
    if (itemDepart->f() != 1) {
        //若不是步行起点，则交换
        itemDepart = (MyItem*)listItem[1];
        itemArrive = (MyItem*)listItem[0];
    }
    //开始组建图元组：用于显示转乘方案的路径
    QGraphicsEllipseItem* myEItem;
    QGraphicsLineItem* myLItem;
    MyScene* scene = (MyScene*)(ui.graphicsView->scene());
    QBrush qbrush(QColor(10, 105, 45));
    //设置颜色
    QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin);
    //站点的笔
    QPen qpenl(qbrush, 3, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin);
```

```
    //线路的笔
    route = true;
    //进入查询路径显示期间，不响应步行起点与终点选取
    int c,n = 0;
    //c 为可行转乘方案数，n 为最少转乘次数
    ROUTE r[100];
    //一次查询，最多返回 100 条可行转乘方案
    for (int d = 0; d < itemDepart->c(); d++) {
        //接近起点的所有公交站点
        int s=(*itemDepart)[d]; //获得起点站点编号 s
        for (int a = 0; a < itemArrive->c(); a++) {
            //接近终点的所有公交站点
            int t = (*itemArrive)[a];
            //获得终点站点编号 t
            if (s == t) {
                //起点站和终点站相同不用转乘
                deleteItems();
                //删除先前的转乘方案路径显示
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6,
6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);
                //步行起点的位置
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(),
itemArrive->x(), itemArrive->y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);
                //到步行终点的路径
                myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6,
6);
                myEItem->setPen(qpens);
                myEItem->setBrush(QBrush(QColor(255, 170, 00)));
                gItem->addToGroup(myEItem);
                //步行终点的位置
                scene->addItem(gItem);
                ui.graphicsView->setScene(scene);
                continue;
            }
            c=MatPathCalc::tra.miniTran(s, t, n, r);
            //得到的转乘方案数
            for (int m = 0; m < c; m++){
                //对于第 m 个转乘方案即 route=r[m]，转乘次数都为 n
                deleteItems();
```

```
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6,
6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);
                //步行起点的位置
                int fr = s, to = t;
                //起始站 fr 与终点站 to
                int fm, tm;
                //已乘线路 fm 与 z 转乘线路 tm
                int bg, ed;
                //线路中的起始站点序号 bg,终止站点序号 ed
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(),
gis->st[s].X(), gis->st[s].Y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);
                //到起点站
                if (n == 1 && r[m][0].S() == -1) {
                    //即从 i 路到 i 路(此时 S()==-1)不用转乘
                    fm = r[m][0].F();
                    //已乘线路序号 fm
                    bg = MatPathCalc::ls[fm].has(fr);
                    //起始站点在线路中的序号
                    ed = MatPathCalc::ls[fm].has(to);
                    //终止站点在线路中的序号
                    if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                    for (int y = bg; y < ed; y++)
                        //从起始站点下一序号到终止站点序号
                    {
                        fr= MatPathCalc::ls[fm][y];
                        //得到该站点序号对应的站点编号
                        to= MatPathCalc::ls[fm][y + 1];
                        //得到该站点序号对应的站点编号
                        myLItem = new QGraphicsLineItem(MatPathCalc::st[fr].X(),
MatPathCalc::st[fr].Y(), MatPathCalc::st[to].X(), MatPathCalc::st[to].Y());
                        myLItem->setPen(qpenl);
                        gItem->addToGroup(myLItem);
                        //到下一站的路径
                    }
                }
                else {
                    for (int y = 0; y < n; y++)
                        //对于每个转乘
                    {
```

```
                    fm = r[m][y].F();
                    //对于每个转乘的起始线路
                    bg = MatPathCalc::ls[fm].has(fr);
                    to = r[m][y].S();
                    //对于起始线路的转乘站点
                    ed = MatPathCalc::ls[fm].has(to);
                    if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                    for (int u = bg; u < ed; u++)
                        //从起始站点下一序号到终止站点序号
                    {
                        int ff = MatPathCalc::ls[fm][u];
                        //得到该站点序号对应的站点编号
                        int tt = MatPathCalc::ls[fm][u+1];
                        //得到该站点序号对应的站点编号
                        myLItem = new QGraphicsLineItem(MatPathCalc::st[ff].X(),
MatPathCalc::st[ff].Y(), MatPathCalc::st[tt].X(), MatPathCalc::st[tt].Y());
                        myLItem->setPen(qpenl);
                        gItem->addToGroup(myLItem);
                        //到下一站的路径
                    }
                    fr = to;
                    //作为下一起点
                }
                fm = r[m][n-1].T();
                //对于最后乘坐的线路
                bg = MatPathCalc::ls[fm].has(fr);
                ed = MatPathCalc::ls[fm].has(t);
                if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                for (int u = bg; u < ed; u++)
                    //从起始站点下一序号到终止站点序号
                {
                    int ff = MatPathCalc::ls[fm][u];
                    //得到该站点序号对应的站点编号
                    int tt = MatPathCalc::ls[fm][u + 1];
                    //得到该站点序号对应的站点编号
                    myLItem = new QGraphicsLineItem(MatPathCalc::st[ff].X(),
MatPathCalc::st[ff].Y(), MatPathCalc::st[tt].X(), MatPathCalc::st[tt].Y());
                    myLItem->setPen(qpenl);
                    gItem->addToGroup(myLItem);
                    //到下一站的路径
                }
            }
            myLItem = new QGraphicsLineItem(MatPathCalc::st[t].X(),
MatPathCalc::st[t].Y(), itemArrive->x(), itemArrive->y());
            myLItem->setPen(qpenl);
```

```
                gItem->addToGroup(myLItem);
                //到步行终点的路径
                myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6,
6);
                myEItem->setPen(qpens);
                myEItem->setBrush(QBrush(QColor(255, 170, 00)));
                gItem->addToGroup(myEItem);
                //步行终点的位置
                scene->addItem(gItem);
                ui.graphicsView->setScene(scene);
            }
        }
    }
    this->m_Timer->start(5000);
    //展示查询的路径结果 5 秒
}
void Cpp_ex5_hust_navi::zdjl() {
    //先计算最短距离的路径,先获得起点坐标和终点坐标
    if (ui.action_zdjl->isChecked() == false) return;
    //鼠标点击一次触发两次，第二次出发直接返回
    ui.action_zdjl->setChecked(false);
    //鼠标第一次出发，设置状态为 false,防止第 2 次出发进入
    QList<QGraphicsItem*> listItem;
    if ((depart && arrive) == false) return;
    listItem = ui.graphicsView->scene()->items();
    MyItem* itemDepart = (MyItem*)listItem[0];
    MyItem* itemArrive = (MyItem*)listItem[1];
    if (itemDepart->f() != 1) {
        //若不是起点，则交换
        itemDepart = (MyItem*)listItem[1];
        itemArrive = (MyItem*)listItem[0];
    }
    //开始组建图元组：用于显示转乘方案的路径
    QGraphicsEllipseItem* myEItem;
    QGraphicsLineItem* myLItem;
    MyScene* scene = (MyScene*)(ui.graphicsView->scene());
    QBrush qbrush(QColor(210, 0, 160));
    //设置颜色
    QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin);
    //站点的笔
    QPen qpenl(qbrush, 3, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin);
    //线路的笔
    route = true;
    //进入查询路径显示期间，不响应步行起点与终点选取
    int c = 0;
```

```
    //c 为可行线路数
    double dist = 0;
    //dist 为最短距离
    ROUTE r[100];
    for (int d = 0; d < itemDepart->c(); d++) {
        //接近起点的所有公交站点
        int s = (*itemDepart)[d];
        //获得起点站点编号 s
        for (int a = 0; a < itemArrive->c(); a++) {
            //接近终点的所有公交站点
            int t = (*itemArrive)[a];
            //获得终点站点编号 t
            if (s == t) {
                //起点站和终点站相同不用转乘
                deleteItems();
                //删除先前的转乘方案路径显示
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6,
6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);
                //步行起点的位置显示
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(),
itemArrive->x(), itemArrive->y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);
                //到步行终点的路径
                myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6,
6);
                myEItem->setPen(qpens);
                myEItem->setBrush(QBrush(QColor(255, 20, 160)));
                gItem->addToGroup(myEItem);
                //步行终点的位置
                scene->addItem(gItem);
                ui.graphicsView->setScene(scene);
                continue;
            }
            c = MatPathCalc::tra.miniDist(s, t, dist, r);
            //得到的转乘方案数 c
            for (int m = 0; m < c; m++) {
                //对于第 m 个转乘方案即 route=r[m],最短距离都为 dist
                deleteItems();
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6,
```

```
6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);
                //步行起点的位置显示
                int fr = s, to = t;
                //起始站 fr 与终点站 to
                int fm, tm;
                //已乘线路 fm 与 z 转乘线路 tm
                int bg, ed;
                //线路中的起始站点序号 bg,终止站点序号 ed
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(),
gis->st[s].X(), gis->st[s].Y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);
                //到起点站的路径显示
                int n = r[m];
                //当前 route 中的转乘次数
                if (n == 1 && r[m][0].S()==-1){
                    //即从 i 路到 i 路(此时 S()==-1)不用转乘
                    fm = r[m][0].F();
                    //已乘线路序号 fm
                    bg = MatPathCalc::ls[fm].has(fr);
                    //起始站点在线路中的序号
                    ed = MatPathCalc::ls[fm].has(to);
                    //终止站点在线路中的序号
                    if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                    for (int y = bg; y<ed; y++)
                        //从起始站点下一序号到终止站点序号
                    {
                        fr = MatPathCalc::ls[fm][y];
                        //得到该站点序号对应的站点编号
                        to = MatPathCalc::ls[fm][y + 1];
                        //得到该站点序号对应的站点编号
                        myLItem=new QGraphicsLineItem(MatPathCalc::st[fr].X(),
MatPathCalc::st[fr].Y(), MatPathCalc::st[to].X(), MatPathCalc::st[to].Y());
                        myLItem->setPen(qpenl);
                        gItem->addToGroup(myLItem);
                        //到下一站
                    }
                }
                else {
                    for (int y = 0; y < n; y++)
                        //对于每个转乘画出路径
                    {
```

```
                    fm = r[m][y].F();
                    //对于每个转乘的起始线路
                    bg = MatPathCalc::ls[fm].has(fr);
                    to = r[m][y].S();
                    //对于起始线路的转乘站点
                    ed = MatPathCalc::ls[fm].has(to);
                    if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                    for (int u = bg; u < ed; u++)
                        //从起始站点下一序号到终止站点序号
                    {
                        int ff = MatPathCalc::ls[fm][u];
                        //得到该站点序号对应的站点编号
                        int tt = MatPathCalc::ls[fm][u + 1];
                        //得到该站点序号对应的站点编号
                        myLItem = new QGraphicsLineItem(MatPathCalc::st[ff].X(),
MatPathCalc::st[ff].Y(), MatPathCalc::st[tt].X(), MatPathCalc::st[tt].Y());
                        myLItem->setPen(qpenl);
                        gItem->addToGroup(myLItem);
                        //到下一站的路径
                    }
                    fr = to;
                    //作为下一起点
                }
                fm = r[m][n - 1].T();
                //对于最后乘坐的线路
                bg = MatPathCalc::ls[fm].has(fr);
                ed = MatPathCalc::ls[fm].has(t);
                if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                for (int u = bg; u < ed; u++)
                    //从起始站点下一序号到终止站点序号
                {
                    int ff = MatPathCalc::ls[fm][u];
                    //得到该站点序号对应的站点编号
                    int tt = MatPathCalc::ls[fm][u + 1];
                    //得到该站点序号对应的站点编号
                    myLItem = new QGraphicsLineItem(MatPathCalc::st[ff].X(),
MatPathCalc::st[ff].Y(), MatPathCalc::st[tt].X(), MatPathCalc::st[tt].Y());
                    myLItem->setPen(qpenl);
                    gItem->addToGroup(myLItem);
                    //到下一站的路径
                }
            }
            myLItem = new QGraphicsLineItem(MatPathCalc::st[t].X(),
MatPathCalc::st[t].Y(), itemArrive->x(), itemArrive->y());
            myLItem->setPen(qpenl);
```

```
                gItem->addToGroup(myLItem);
                //到步行终点的路径
                myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6,
6);
                myEItem->setPen(qpens);
                myEItem->setBrush(QBrush(QColor(255, 170, 00)));
                gItem->addToGroup(myEItem);
                //步行终点的位置显示
                scene->addItem(gItem);
                ui.graphicsView->setScene(scene);
            }
        }
    }
    this->m_Timer->start(5000);  //展示查询的路径结果 5 秒
}
void Cpp_ex5_hust_navi::search_stop() {
    search_ui->show();
}


/* in File InputWidgets.cpp */

#include <QFileDialog>
#include <QMessageBox>
#include "InputWidgets.h"
#include "logiclayer.h"
#include "Cpp_ex5_hust_navi.h"

//自定义的站点及线路输入界面
InputWidgets::InputWidgets(QWidget *parent)
    : QWidget(parent)
{
    ui.setupUi(this);
    connect(ui.pushButtonStop, SIGNAL(clicked( )), this, SLOT(inputStop()),
Qt::UniqueConnection);
    connect(ui.pushButtonLine, SIGNAL(clicked( )), this, SLOT(inputLine()),
Qt::UniqueConnection);
    connect(ui.pushButtonDone, SIGNAL(clicked()), this, SLOT(checkFile()),
Qt::UniqueConnection);
    connect(ui.pushButtonQuit, SIGNAL(clicked()), this, SLOT(close()),
Qt::UniqueConnection);
}
void InputWidgets::myShow(QGraphicsView* p) {
    parnt = p;
    show();
```

```
}
InputWidgets::~InputWidgets()
{
}
void InputWidgets::inputStop() {
    ui.labelHits->setText("");
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), ".",
tr("*.txt"));
    ui.textEditStop->setText(fileName);
}
void InputWidgets::inputLine() {
    ui.labelHits->setText("");
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), "lines",
tr("*.txt"));
    ui.textEditLine->setText(fileName);
}

void InputWidgets::checkFile() {
    QString  fs = ui.textEditStop->toPlainText();
    QString  fl = ui.textEditLine->toPlainText();
    if (fs.isEmpty() && fl.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromLocal8Bit("操作提示:没有输入站点及线路文件路
径! ")); //不用 fromLocal8Bit 显示乱码
        ui.textEditStop->setFocus();
        return;
    }
    if (fs.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromLocal8Bit("操作提示：没有输入站点文件路径!
"));
        ui.textEditStop->setFocus();
        return;
    }
    if (fl.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromLocal8Bit("操作提示：没有输入线路文件路径!
"));
        ui.textEditLine->setFocus();
        return;
    }
    //处理站点文件
    ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)"); //设置操作提示信息显示颜色
    ui.labelHits->setText(QString::fromLocal8Bit("操作提示：正在处理站点和线路文
件..."));
```

```
    try {//读入站点及线路文件并初始化
        if (gis != nullptr) delete gis;
        gis = new MatPathCalc(fs.toStdString().c_str(), fl.toStdString().c_str());
        ((MyScene*)(parnt->scene()))->stopLines(parnt); //在背景地图上画出站点及公交线
路
    }
    catch (...) {//读入或初始化失败
        gis = nullptr;
        close();
        QMessageBox::information(NULL, QString::fromLocal8Bit("操作提示"),
QString::fromLocal8Bit("公交站点或公交线路文件读入及初始化失败！"));
    }
    ui.labelHits->setText("");
    close();
}


/* in File search_widget.cpp */

#include "search_widget.h"
#include "Cpp_ex5_hust_navi.h"
#include "logiclayer.h"
#include "InputWidgets.h"
#include <QDialog>
#include <QDesktopWidget>
#include <QApplication>
#include <QMessageBox>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsEllipseItem>
#include "SuspendForm.h"
#include "search_widget.h"

search_widget::search_widget(QWidget *parent)
    : QWidget(parent), pos_x(-1), pos_y(-1)
{
    ui.setupUi(this);

    // 读取数据库文件
    std::ifstream org_data("d:/organization.txt", std::ios::in);
    if (org_data) {
        std::cout << "初始化站点数据库成功";
        std::string stop_name;
        int x, y;
        while (1) {
            org_data >> stop_name;
```

```
            if (org_data.fail()) break;
            org_data >> x >> y;
            data.push_back(sp{ stop_name, x, y });
        }
        std::cout << "...完成中文模糊查找模块初始化" << std::endl;
    }
    else {
        std::cout << "读取文件失败！" << std::endl;
        throw "init file searcher error!";
    }
}


int search_widget::get_x() {
    return this->pos_x;
}

int search_widget::get_y() {
    return this->pos_y;
}

int search_widget::fuzzy_match(std::string target) {
    int max_n = -1, best_ind = -1;
    int cur_n;
    for (int i = 0; i < data.size(); i++) {
        cur_n = 0;
        for (int j = 0; j < target.size(); j++) {
            if (data[i].name.find(target[j]) != std::string::npos) {
                cur_n++;
            }
        }
        if (cur_n > max_n) {
            max_n = cur_n;
            best_ind = i;
        }
    }
    return best_ind;
}

void search_widget::search() {
    std::string stop_name = ui.text_search->toPlainText().toStdString();  // 获取查找
名
    int best_match_ind = fuzzy_match(stop_name);
    pos_x = data[best_match_ind].x;
    pos_y = data[best_match_ind].y;
```

```
}

void search_widget::set_start() {
    search();
    //depart = true;              // 设为起点
    this->close();  // 查找完后关闭搜索框
}

void search_widget::set_end() {
    search();
    //arrive = true;              // 设为终点
    this->close();  // 查找完后关闭搜索框
}


/* in File logiclayer.h */

#pragma once

#define NULL 0

class STOP {        // 描述一个公交站点
    int index;  // 站点编号,从0开始
    int x, y;   // 站点坐标
public:
    STOP(int n = 0, int x = 0, int y = 0);
    virtual int& X();
    virtual int& Y();
    virtual int& N();
};
class LINE {    // 描述一条公交线路
    const int index;    // 公交线路从1开始计算
    int* const stop;    // 站点编号
    const int nofs;     // 站点数量
public:
    LINE(int index = 0, int nofs = 0, int* stop = NULL);
    LINE(const LINE& r);
    LINE(LINE&& r) noexcept;
    LINE& operator=(const LINE& r);
    LINE& operator=(LINE&& r) noexcept;
    virtual int has(int s) const;   // 是否含有站点s，-1表示不包含
    virtual int cross(const LINE& b) const;     // 判断两条公交线路是否相交，若是，则返
回1
    virtual operator int()const;    // 取公交线路编号
    virtual int NOFS() const;       // 取公交线路的站点数量
```

```
    virtual double dist(int b, int e)const;     // 取线路从站次 d 到站次 a 之间的距离
    virtual int& operator[](int x); // 取线路某个站次的站点编号
    virtual ~LINE() noexcept;
};

class TRAN {        // 从线路 from 经过 stop 转到 to
    int from;
    int to;
    int stop;
public:
    TRAN(int from = 0, int to = 0, int stop = 0);
    int operator==(const TRAN& t) const;
    virtual int& F();       // 取 from
    virtual int& T();       // 取 to
    virtual int& S();       // 取 stop
};

class ROUTE {       //一个转乘路径
    TRAN*const tran;//转乘路径上的所有转乘
    const int noft; //转乘路径上转乘次数
public:
    ROUTE(TRAN* tran = NULL, int noft = 0);
    ROUTE(const TRAN& t);
    ROUTE(const ROUTE& r);
    ROUTE(ROUTE&& r)noexcept;
    virtual operator int()const;                    // 取转乘次数
    virtual int operator==(const ROUTE& r)const;
    virtual ROUTE operator *()const;                // 要化简
    virtual TRAN& operator[](int);                  // 路径上所有的转乘站点
    virtual ROUTE operator+(const ROUTE& r)const;   // 转乘路径连接
    virtual ROUTE& operator=(const ROUTE& r);
    virtual ROUTE& operator=(ROUTE&& r) noexcept;
    virtual ROUTE& operator+=(const ROUTE& r);
    virtual ~ROUTE() noexcept;
    virtual int print()const;
};

class NODE {                    // 矩阵，记录转乘次数与线路
    ROUTE* const p;
    int n;
public:
    NODE(ROUTE* p, int n);
    NODE(int n = 0);
    NODE(const NODE& n);
    NODE(NODE&& n) noexcept;
```

```
    virtual NODE operator*()const;                  //去除矩阵的环
    virtual NODE operator+(const ROUTE& n)const;    //元素增加路径
    virtual NODE operator+(const NODE& n)const;
    virtual NODE operator*(const NODE& n)const;     //元素路径转乘连接
    virtual NODE& operator=(const NODE& n);
    virtual NODE& operator+=(const NODE& n);
    virtual NODE& operator+=(const ROUTE& n);
    virtual NODE& operator*=(const NODE& n);
    virtual NODE& operator=(NODE&& n)noexcept;
    virtual ROUTE& operator[](int x);               //获得第 x 个转乘路径
    virtual operator int& ();                       //返回可转乘路径的数量
    virtual ~NODE()noexcept;
    virtual void print()const;
};

class TMAP {
    NODE* const p;          //指向闭包矩阵的 r*c 个元素
    const int r, c;         //闭包矩阵的行列数
public:
    TMAP(int r = 0, int c = 0);
    TMAP(const TMAP& a);
    TMAP(TMAP&& a)noexcept;
    virtual ~TMAP();
    virtual int notZero()const;
    virtual int miniTran(int b, int e, int& noft, ROUTE(&r)[100])const;
    virtual int miniDist(int b, int e, double& dist, ROUTE(&r)[100])const;
    static double getDist(int b, int e, ROUTE& r);
    virtual NODE* operator[](int r);
    virtual int& operator()(int r, int c);
    virtual TMAP operator*(const TMAP& a)const;
    virtual TMAP operator+(const TMAP& a)const;
    virtual TMAP& operator=(const TMAP& a);
    virtual TMAP& operator=(TMAP&& a);
    virtual TMAP& operator+=(const TMAP& a);
    virtual TMAP& operator*=(const TMAP& a);
    virtual TMAP& operator()(int r, int c, const ROUTE& a);
    virtual void print() const;
};
struct MatPathCalc {
    static STOP* st;        // 所有的公交站点
    static LINE* ls;        // 所有的公交线路
    static int ns, nl;      // 公交站数和公交线路数
    static TMAP raw, tra;   // 原始转乘矩阵、闭包转乘矩阵
    static int obs;         // 对象数量
public:
```

```
    MatPathCalc();
    MatPathCalc(const char* flstop, const char* flline);    // 用文件加载地图
    int miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n, ROUTE(&r)[100]);
    int miniDist(int fx, int fy, int tx, int ty, int& f, int& t, double& d, ROUTE(&r)[100]);
    ~MatPathCalc();
};
extern MatPathCalc* gis;


/* in File logiclayer.cpp */

#define _CRT_SECURE_NO_WARNINGS
#include "logiclayer.h"
#include <math.h>
#include <cstdio>


STOP* MatPathCalc::st = 0;
LINE* MatPathCalc::ls = 0;
int MatPathCalc::ns = 0;
int MatPathCalc::nl = 0;
int MatPathCalc::obs = 0;
TMAP MatPathCalc::raw;
TMAP MatPathCalc::tra;

/* STOP 类函数定义部分 */

STOP::STOP(int n, int x, int y) :index(n), x(x), y(y) {}

int& STOP::X() {
    return x;
}

int& STOP::Y() {
    return y;
}

int& STOP::N() {
    return index;
}



/* LINE 类函数定义部分 */
```

```
LINE::LINE(int index, int nofs, int* stop) :index(index), stop(nofs ? new int[nofs] :
0), nofs(stop ? nofs : 0) {
    if (LINE::stop == NULL && this->nofs != 0) throw "Memory allocation error in class
LINE!";
    for (int i = 0; i < nofs; ++i) LINE::stop[i] = stop[i];
}

LINE::LINE(const LINE& r) :index(r.index), stop(r.nofs ? new int[r.nofs] : NULL),
nofs(stop ? r.nofs : 0) {
    if (stop == NULL && r.nofs != 0) throw "Memory allocation error in class LINE!";
    for (int i = 0; i < nofs; ++i) stop[i] = r.stop[i];
}

LINE::LINE(LINE&& r) noexcept :index(r.index), stop(r.stop), nofs(r.nofs) {
    *(int*)&(r.index) = 0;
    *(int*)&(r.nofs) = 0;
    *(int**)&(r.stop) = 0;
}

LINE& LINE::operator=(const LINE& r) {
    if (this == &r) return *this;
    if (stop) delete[] stop;
    *(int*)&(r.index) = r.index;
    *(int**)&(r.stop) = new int[r.nofs];
    if (stop == NULL) throw "ERROR in operator = of class Line";
    *(int*)&(r.nofs) = stop ? r.nofs : 0;
    for (int i = 0; i < nofs; ++i) stop[i] = r.stop[i];
    return *this;
}

LINE& LINE::operator=(LINE&& r) noexcept {
    if (this == &r) return *this;
    if (stop) delete[] stop;
    *(int*)&(index) = r.index;
    *(int*)&(nofs) = r.nofs;
    *(int**)&(stop) = r.stop;
    *(int*)&(r.index) = 0;
    *(int*)&(r.nofs) = 0;
    *(int**)&(r.stop) = 0;
    return *this;
}

// 是否含有站点 s，-1 表示不包含
int LINE::has(int s) const {
    for (int i = 0; i < nofs; ++i)
```

```
        if (stop[i] == s) return i;
    return -1;
}

// 判断两条公交线路是否相交，若是，则返回 1
int LINE::cross(const LINE& b) const {
    if (this == &b) return 0;
    for (int i = 0; i < nofs; ++i)
        if (b.has(stop[i]) != -1) return 1;
    return 0;
}

// 取公交线路编号
LINE::operator int()const {
    return index;
}

// 取公交线路的站点数量
int LINE::NOFS() const {
    return nofs;
}

// 取线路从站次 d 到站次 a 之间的距离
double LINE::dist(int b, int e)const {
    double d = 0.0; // d 是站点之间的距离
    int b_i, e_i;   // b_i 是 b 对应的序列编号，e_i 是 e 对应的序列编号
    if ((b_i = has(b)) == -1) throw "Station not found when calculating distances!";
    if ((e_i = has(e)) == -1) throw "Station not found when calculating distances!";
    if (b_i > e_i) {
        // 若 b 比 e 大，则交换
        b = b_i;
        b_i = e_i;
        e_i = b;
    }
    int x_1, y_1, x_2, y_2;
    for (int i = b_i; i < e_i; ++i) {
        x_1 = MatPathCalc::st[stop[i]].X();
        y_1 = MatPathCalc::st[stop[i]].Y();
        x_2 = MatPathCalc::st[stop[i + 1]].X();
        y_2 = MatPathCalc::st[stop[i + 1]].Y();
        d += sqrt(double((x_1 - x_2) * (x_1 - x_2) + (y_1 - y_2) * (y_1 - y_2)));
    }
    return d;
}
```

```
// 取线路某个站次的站点编号
int& LINE::operator[](int x) {
    return stop[x];
}

LINE::~LINE() noexcept {
    if (stop) {
        delete[]stop;
        *(int**)&stop = NULL;
        *(int*)&index = 0;
        *(int*)&nofs = 0;
    }
}

/* TRAN 类函数定义部分 */

TRAN::TRAN(int from, int to, int stop) :from(from), to(to), stop(stop) {}

int TRAN::operator==(const TRAN& t) const {
    return from == t.from && to == t.to && stop == t.stop;
}

// 取 from
int& TRAN::F() {
    return from;
}

// 取 to
int& TRAN::T() {
    return to;
}

// 取 stop
int& TRAN::S() {
    return stop;
}


/* ROUTE 类函数定义部分 */

ROUTE::ROUTE(TRAN* tran, int noft) :tran(noft ? new TRAN[noft] : NULL), noft(tran ? noft :
0) {
    if (ROUTE::tran == NULL && noft != 0) throw "Memory allocation error when constructing
class ROUTE!";
    for (int i = 0; i < noft; ++i) ROUTE::tran[i] = tran[i];
```

```
}

ROUTE::ROUTE(const TRAN& t) :tran(new TRAN[1]), noft(tran ? 1 : 0) {
    if (tran == NULL) throw "error!";
    if (tran) *tran = t;
}

ROUTE::ROUTE(const ROUTE& r) :tran(r.noft ? new TRAN[r.noft] : NULL), noft(tran ? r.noft :
0) {
    if (tran == NULL && r.noft != 0) throw "Memory allocation error when constructing
class ROUTE!";
    for (int i = 0; i < noft; ++i) tran[i] = r.tran[i];
}

ROUTE::ROUTE(ROUTE&& r)noexcept :tran(r.tran), noft(r.noft) {
    *(TRAN**)&(r.tran) = NULL;
    *(int*)&(r.noft) = 0;
}

// 取转乘次数
ROUTE::operator int()const {
    return noft;
}

int ROUTE::operator==(const ROUTE& r)const {
    if (noft != r.noft) return 0;
    int flag = 1;   // flag为比较结果
    for (int i = 0; i < noft; ++i) {
        if (!(tran[i] == r.tran[i])) {
            flag = 0;
            break;
        }
    }
    return flag;
}

// 要化简，去掉重复转乘
ROUTE ROUTE::operator *()const {
    TRAN* t = new TRAN[noft];
    if (t == NULL) throw "Error in ROUTE operator *()!";
    int nn = noft;
    for (int x = 0; x < nn; x++) t[x] = tran[x];
    for (int x = 0; x < nn - 1; ++x)
        for (int y = x + 1; y < nn; ++y) {
            if (t[x].S() == t[y].S() && t[x].F() == t[y].T()) {
```

```
                // 转乘重复
                for (int m = x, n = y + 1; n < nn; n++, m++) t[m] = t[n];
                nn -= y + 1 - x;
                y = x;
            }
        }
    ROUTE r(t, nn);
    delete[] t;
    return r;
}

// 路径上所有的转乘站点
TRAN& ROUTE::operator[](int x) {
    if (x<0 || x>noft) throw "ROUTE error!";
    return tran[x];
}

// 转乘路径连接
ROUTE ROUTE::operator+(const ROUTE& r)const {
    if (noft == 0) return *this;
    if (r.noft == 0) return r;
    // 当两个 ROUTE 均非空
    ROUTE s;
    if (tran[noft - 1].T() != r.tran[0].F()) throw "Route cannot be connected.";
    try {
        *(TRAN**)&(s.tran) = new TRAN[noft + r.noft];
    }
    catch (...) {
        throw "Memory allocation in constructing TRAN by ROUTE.";
    }
    (int&)(s.noft) = s.tran ? noft + r.noft : 0;
    int x = 0;
    for (; x < noft; ++x) s.tran[x] = tran[x];
    for (int i = 0; i < r.noft; ++i) s.tran[x++] = r.tran[i];
    return *s;
}

ROUTE& ROUTE::operator=(const ROUTE& r) {
    if (this == &r) return *this;
    if (tran) delete[] tran;
    try {
        (TRAN*&)tran = new TRAN[r.noft];
    }
    catch (...) {
        throw "Memory allocation in constructing TRAN by ROUTE.";
```

```
    }
    (int&)noft = tran ? r.noft : 0;
    for (int x = 0; x < noft; ++x) tran[x] = r.tran[x];
    return *this;
}

ROUTE& ROUTE::operator=(ROUTE&& r) noexcept {
    if (this == &r) return *this;
    if (tran) delete[] tran;
    (TRAN*&)tran = r.tran;
    (int&)noft = r.noft;
    (TRAN*&)(r.tran) = NULL;
    (int&)(r.noft) = 0;
    return *this;
}

ROUTE& ROUTE::operator+=(const ROUTE& r) {
    return *this = *this + r;
}

ROUTE::~ROUTE() noexcept {
    if (tran) {
        delete[] tran;
        *(TRAN**)&(tran) = NULL;
        *(int*)&(noft) = 0;
    }
}

// 打印路径
int ROUTE::print()const {
    for (int i = 0; i < noft; ++i)
        if (tran[i].S() == -1) printf("Arrive by bus #%d.\n", tran[i].F() + 1);
        else printf("Tranship from bus #%d to bus #%d at stop #%d.", tran[i].F() + 1,
tran[i].T() + 1, tran[i].S() + 1);
    return noft;
}


//=====================================NODE=======================================
=
NODE::NODE(ROUTE* p, int n):p(n?new ROUTE[n]:nullptr), n(p?n:0) {
    if (NODE::p == nullptr && n!=0) throw "memory allocation for ROUTE construction
error!";
    for (int x = 0; x < n; x++) NODE::p[x] = p[x];
}
```

```
NODE::NODE(int n):p(n?new ROUTE[n]:nullptr), n(p?n:0) {
    if (p == nullptr && n!=0) throw "memory allocation for NODE construction error!";
}
NODE::NODE(const NODE& n) : p(n.n ? new ROUTE[n.n] : nullptr), n(p ? n.n : 0) {
    if (p == nullptr && n.n!=0) throw "memory allocation for NODE construction error!";
    for (int x = 0; x<NODE::n; x++) p[x] = n.p[x];
}
NODE::NODE(NODE&& n)noexcept: p(n.p), n(n.n) {
    (ROUTE*&)(n.p) = nullptr;
    (int&)n.n = 0;
}
NODE  NODE::operator*()const {
    int n = NODE::n;
    if (n == 0) return *this;
    ROUTE* t = new ROUTE[n];
    if (t== nullptr) throw "memory allocation for NODE::operator*() error!";
    for (int x = 0; x < n; x++) t[x] = p[x];
    for (int x = 0; x < n - 1; x++)
        for (int y = x + 1; y < n; y++) {
            if (t[x]== t[y]) {
                for (int m = x + 1, n = y + 1; n < n - 1; n++, m++)
                    t[m] = t[n];
                n -= (y - x);
                y = x;
            }
        }
    NODE r(t, n);
    try {
        if (t != nullptr)delete[]t;
        t = nullptr;
    }
    catch (...) {
        throw "initializing failed! ";
    }
    return r;
}
NODE  NODE::operator+(const ROUTE& n)const {
    NODE r(NODE::n + 1);
    for (int x = 0; x < NODE::n; x++) r.p[x] = *p[x];
    r.p[NODE::n] = n;
    return *r;
}
NODE  NODE::operator+(const NODE& n)const {
    if (NODE::n == 0) return n;
    if (n.n == 0) return *this;
```

```
    NODE r(NODE::n + n.n);
    for (int x = 0; x < NODE::n; x++) r.p[x] = *p[x];
    for (int x = 0; x < n.n; x++) r.p[x+NODE::n] = *n.p[x];
    return *r;
}
NODE  NODE::operator*(const NODE& n)const {
    if (NODE::n == 0) return *this;
    if (n.n == 0) return n;
    NODE r(NODE::n * n.n);
    int m,f,h,k = 0;
    for(int x=0; x< NODE::n; x++)   //当前节点:x=4,y=0,k=4 时异常
        for (int y = 0; y < n.n; y++) {
            if (p[x][-1 + p[x]].T() != n.p[y][0].F()) throw "Can not tansship from
buses!";
            try {
                r.p[k] = p[x] + n.p[y];  //Route 的连接运算
            }
            catch (const char* e) {
                const char* p = e;
            }
            k++;
        }
    return *r;
}
NODE& NODE::operator=(const NODE& n) {
    if(this == &n) return *this;
    if (p) delete[]p;
    (ROUTE*&)p = new ROUTE[n.n];
    if (p == nullptr) throw "Memory allocation for NODE construction error!";
    (int&)(NODE::n) = p ? n.n : 0;
    for (int x= 0; x < n.n; x++) p[x] = n.p[x];
    return *this;
}
NODE& NODE::operator=(NODE&& n)noexcept {
    if (this == &n) return *this;
    if (p) delete[]p;
    (ROUTE*&)p = n.p;
    (int&)(NODE::n) = n.n;
    (ROUTE*&)(n.p) = nullptr;
    (int&)(n.n) = 0;
    return *this;
}
NODE& NODE::operator+=(const ROUTE& n) {
    return *this = *this + n;
}
```

```
NODE& NODE::operator+=(const NODE & n)  {
    return *this = *this + n;
}
NODE& NODE::operator*=(const NODE& n) {
    return *this = *this * n;
}
ROUTE& NODE::operator [](int x) {
    if (x < 0 || x >= n) throw "Subscription x of NODE::operator [](int x) is wrong!";
    return p[x];
}
NODE::operator int&() { return n; }
NODE::~NODE()noexcept {
    if (p) {
        delete[]p;
        (ROUTE*&)p = nullptr;
        (int&)n = 0;
    }
}
void NODE::print()const {
    for (int m = 0; m < n; m++) {
        p[m].print();
    }
}
//==================================TMAP=========================================
==
TMAP::TMAP(int r, int c) : p((r!=0&&c!=0)?new NODE [r * c]:nullptr),  r(p ? r : 0), c(p?
c : 0) {
    if (TMAP::p == nullptr && r != 0 && c != 0) throw "Memory allocation for TMAP
construction error!";
}
TMAP::TMAP(const TMAP& a) : p((a.r!=0 && a.c!=0)?new NODE [a.r*a.c]:nullptr), r(p? a.r:0),
c(p ? a.c:0) {
    if (p == nullptr && a.r != 0 && a.c != 0) throw "Memory allocation for TMAP construction
error!";
    for (int k = r * c - 1; k >= 0; k--) p[k] = a.p[k];
}
TMAP::TMAP(TMAP&& a)noexcept: p(a.p), r(a.r), c(a.c){
    (NODE*&)(a.p) = nullptr;
    (int&)(a.r) = (int&)(a.c) = 0;
}
TMAP::~TMAP() {
    if (p) {
        delete[] p;
        (NODE*&)p = nullptr;
        (int&)r = (int&)c = 0;
```

```
    }
}
int TMAP::notZero()const {
    for (int x = r * c - 1; x >= 0; x--) if (p[x].operator int &()==0) return 0;
    return 1;
}
int& TMAP::operator()(int x, int y) {
    if (x < 0 || x >= r) throw "Subscript bound error!";
    if (y < 0 || y >= c) throw "Subscript bound error!";
    return p[x * c + y];
}
NODE* TMAP::operator[](int r) {
    if (r < 0 || r >= TMAP::r) throw "Subscript bound error!";
    return p+r * c;
}
TMAP& TMAP::operator=(const TMAP& a) {
    if (this == &a) return *this;
    if (p) delete[]p;
    (NODE*&)p = new NODE[a.r*a.c];
    if (p == nullptr) throw "Memory allocation for TMAP assignment error!";
    (int&)r = a.r;
    (int&)c = a.c;
    for (int k = r * c - 1; k >= 0; k--)
        p[k] = a.p[k];
    return *this;
}
TMAP& TMAP::operator=(TMAP&& a) {
    if (this == &a) return *this;
    if (p) delete[]p;
    (NODE*&)p = a.p;
    (int&)r = a.r;
    (int&)c = a.c;
    (NODE*&)(a.p) = nullptr;
    (int&)(a.r)=(int&)(a.c)=0;
    return *this;
}
TMAP TMAP::operator*(const TMAP& a)const {
    if (c != a.r) throw "TMAP can nott mutiply!";
    int t, m, u, v, w, x, y, z;
    TMAP s(r, a.c);                //每个节点皆为空线路
    for (int h = 0; h < r; h++)
        for (int j = a.c - 1; j >= 0; j--) {
            if (h == j) continue;
            t = h * s.c + j;
            //s.p[t] = NODE();      //原有路线数:方阵
```

```
            for (int k = 0; k < c; k++)
                if (k != h && k!=j) //新增路线数
                    s.p[t]+= p[h * c + k] * a.p[k * a.c + j];
        }
    return s;
}
TMAP TMAP::operator+(const TMAP& a)const {
    if (r!=a.r && c !=a.c ) throw "TMAP can nott add!";
    TMAP s(*this);
    for (int h = r * c - 1; h >= 0; h--) s.p[h] += a.p[h];
    return s;
}
TMAP& TMAP::operator+=(const TMAP& a) { return *this = *this + a; }
TMAP& TMAP::operator*=(const TMAP& a) { return *this = *this * a; }
TMAP& TMAP::operator()(int ro, int co, const ROUTE& a) {
    p[ro * c + co] += a;
    return *this;
}
//根据起点站 s 和终点站 t 找到最少转乘次数 noft 的若干线路 r,返回线路数
int TMAP::miniTran(int s, int t, int& notf, ROUTE(&r)[100])const {
    int k, u, v, w, x, y, z;//z:返回实际最少转乘线路数
    int b = 0, e = 0;       //包含起点 s 的起始线路数 b(线路存放在 bls),包含终点 t 的起始线路数 e(线路存放在 els)
    int nott[100]{};        //对应规划线路 r 的转乘次数
    int bls[20], els[20];   //bls:包含起点 s 的起始线路
    NODE rou;
    for (z = 0; z < MatPathCalc::nl; z++) { //寻找包含起点或终点相关公交线路下标
        if (MatPathCalc::ls[z].has(s) != -1) if (b < 20) bls[b++] = z;
        if (MatPathCalc::ls[z].has(t) != -1) if (e < 20) els[e++] = z;
    }
    for (x = z = 0; x < b; x++)
        for (y = 0; y < e; y++) {
            rou = MatPathCalc::tra[bls[x]][els[y]]; //得到两线路的所有转乘线路
            w = MatPathCalc::tra(bls[x], els[y]);
            if (w == 0) continue;   //转乘线路数==0
            for (v = 0; v < w; v++) {
                u = rou[v];         //线路得到转乘次数
                if (z == 0 || u < nott[0]) {//若 nott 为空，或转乘次数比 nott[0]小
                    nott[0] = u;
                    r[0] = rou[v];
                    z = 1;
                }
                if (u == nott[0]) { //和已有线路转乘次数相同时，则插入
                    if (z == 100) return z;
                    nott[z] = u;
```

```
                    r[z] = rou[v];
                    z++;
                }
            }
        }
    notf = nott[0]; //nott[0]到的 nott[z-1]的转乘次数都相同
    return z;       //返回最少转乘线路数
}
//起站 b,终站 e,使用路线 r 的距离
double TMAP::getDist(int b, int e, ROUTE& r) {
    int x, y;
    double d = 0;
    if (1 == r && r[0].F() == r[0].T()) {   //乘坐线路=转乘线路：不用转乘
        d = MatPathCalc::ls[r[0].F()].dist(b, e);
        return d;
    }
    d = MatPathCalc::ls[r[0].F()].dist(b, r[0].S());
    y = (int)r - 1;         //转乘次数
    for (x = 0; x < y; x++)
        d += MatPathCalc::ls[r[x].T()].dist(r[x].S(), r[x + 1].S());
    d += MatPathCalc::ls[r[y].T()].dist(r[y].S(), e);
    return d;
}
//根据起点站 s 和终点站 t 找到最短距离 dist 若干线路 r，返回实际线路数
int TMAP::miniDist(int s, int t, double& dist, ROUTE(&r)[100])const {
    int k, u, v, w, x, y, z;
    int b = 0, e = 0;       //s:包含站点 s(站点数组下标）的起始线路（存放在 bls)数
    double dot[100]{};      //对应 r 的转乘距离
    int bls[20], els[20];   //e:包含站点 t(站点数组下标）的起始线路（存放在 els)数
    NODE rou;
    for (z = 0; z < MatPathCalc::nl; z++) {            //寻找相关公交线路下标
        if (MatPathCalc::ls[z].has(s) != -1) if (b < 20) bls[b++] = z;
        if (MatPathCalc::ls[z].has(t) != -1) if (e < 20) els[e++] = z;
    }
    for (x = z = 0; x < b; x++)                 //b 个起始站点
        for (y = 0; y < e; y++) {               //e 个到达站点
            rou = MatPathCalc::tra[bls[x]][els[y]];     //得到两线路的所有转乘线路
            w = MatPathCalc::tra(bls[x], els[y]);       //得到两线路的所有转乘线路数
            if (w == 0) continue;               //本次起始站点的转乘线路数 w==0
            for (v=0; v < w; v++) {
                u = TMAP::getDist(s, t, rou[v]);//得到转乘距离
                if (z == 0 || u < dot[0]) {     //若 dot 为空，或转乘次数比 dot[0]小
                    dot[0] = u;
                    r[0] = rou[v];
                    z = 1;
```

```
                }
                if (u == dot[0]) {              //和已有线路转乘距离相同时，则插入
                    if (z == 100) return z;
                    dot[z] = u;
                    r[z] = rou[v];
                    z++;
                }
            }
        }
    dist = dot[0];  //升序插入:dot[0]到 dot[z-1]存储的距离相同
    return z;       //z 个乘坐方案
}

void TMAP::print()const {
    for(int x=0; x<r; x++)
        for (int y = 0; y < c; y++) {
            printf("Node(%d,%d) has %d routes:\n", x, y, (int)(p[x * c + y]));
            p[x * c + y].print();
        }

}
//====================================MatPathCalc================================
======================
MatPathCalc::MatPathCalc() { obs++; }
MatPathCalc::MatPathCalc(const char* flstop, const char* flline) {
    int  m, n, p, q, r, * s, * t;
    FILE* fs, * fl;
    fs = fopen(flstop, "r");
    fl = fopen(flline, "r");
    if (fs == 0 || fl == 0) throw "File open error!";
    fscanf(fs, "%d", &ns);
    st = new STOP[ns];
    for (m = 0; m < ns; m++) {
        fscanf(fs, "%d%d", &st[m].X(), &st[m].Y());
        st[m].N() = m + 1;      //公交线路编号从 1 开始
    }
    fclose(fs);
    fscanf(fl, "%d", &nl);
    s = new int[nl];            //每条线路的站点数
    t = new int[100];           //每条线路的站点数不超过 100 站
    for (m = 0; m < nl; m++) {
        fscanf(fl, "%d", &s[m]);
    }
    *(LINE**)&ls = new LINE[nl];
    for (m = 0; m < nl; m++) {
```

```
    for (n = 0; n < s[m]; n++) {
        fscanf(fl, "%d", &t[n]);
        t[n]--;
    }
    ls[m] = LINE(m + 1, s[m], t);
}
fclose(fl);
for (m = 0; m < nl; m++) {  //构造 raw
    for (p = n = 0; n < nl; n++)
        if (m != n) p += MatPathCalc::ls[m].cross(MatPathCalc::ls[n]);
    if (p == 0) {
        printf("line %d does nott cross any line\n", m + 1);
        throw  "there is independent line";
    }
}
TMAP ra(nl, nl);
ROUTE a;
TRAN* u = new TRAN[100];
for (m = 0; m < nl; m++)
    for (n = 0; n < nl; n++)
    {
        if (m == n) {              //本线路自身可在任一点转移
            u[0] = TRAN(m, n, -1);  //本线路自身可在任一点转移
            a = ROUTE(&u[0],1 );    //线路只有一次转乘
            ra(m, n, a);
            continue;
        }
        p = 0;                     //公交线路交点个数
        for (q = MatPathCalc::ls[m].NOFS() - 1; q >= 0; q--) {
            r = MatPathCalc::ls[m][q];
            if (MatPathCalc::ls[n].has(r) != -1)
            {//每个交点是一个转乘方式：新路线
                u[p] = TRAN(m, n, r);
                a = ROUTE(&u[p++],1); //线路只有一次转乘
                ra(m, n, a);
            }
        }
    }
tra = raw = ra;
for (n = 2; n < nl; n++) {//构造闭包
    raw *= ra;
    tra += raw;
}
raw = ra;
delete s;
```

```
    delete t;
    delete[]u;
    obs++;
}
MatPathCalc::~MatPathCalc() {
    obs--;
    if (obs) return;
    if (st) { delete[]st; *(STOP**)&st = 0; }
    if (ls) { delete[]ls; *(LINE**)&ls = 0; }
}
//根据步行起点和终点找到最少转乘次数 n 的若干线路 r，返回线路数
int MatPathCalc::miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n,
ROUTE(&r)[100]) {
    int m;
    double df, tf, dt, tt;
    f = 0;      //设离起点最近的站点 f
    df = sqrt((st[0].X() - fx) * (st[0].X() - fx) + (st[0].Y() - fy) * (st[0].Y() - fy));
    t = 0;      //设离终点最近的站点 t
    dt = sqrt((st[0].X() - tx) * (st[0].X() - tx) + (st[0].Y() - ty) * (st[0].Y() - ty));
    for (m = 1; m < MatPathCalc::ns; m++) {         //搜索离起点或终点最近的站点
        tf = sqrt((st[m].X() - fx) * (st[m].X() - fx) + (st[m].Y() - fy) * (st[m].Y()
- fy));
        if (df > tf) { df = tf;  f = m; }   //离步行起点最近的站点，存在 f 中
        tt = sqrt((st[m].X() - tx) * (st[m].X() - tx) + (st[m].Y() - ty) * (st[m].Y()
- ty));
        if (dt > tt) { dt = tt;  t = m; }
    }
    if (f == t) return 0;   //起点和终点相同，不用乘车
    return MatPathCalc::tra.miniTran(f, t, n, r);
}
//根据步行起点和终点找到最短距离 d 的若干线路 r,返回线路数
int MatPathCalc::miniDist(int fx, int fy, int tx, int ty, int& f, int& t, double& d,
ROUTE(&r)[100]) {
    int m;
    double df, tf, dt, tt;
    f = 0;
    df = sqrt((st[0].X() - fx) * (st[0].X() - fx) + (st[0].Y() - fy) * (st[0].Y() - fy));
    t = 0;
    dt = sqrt((st[0].X() - tx) * (st[0].X() - tx) + (st[0].Y() - ty) * (st[0].Y() - ty));
    for (m = 1; m < MatPathCalc::ns; m++) {
        tf = sqrt((st[m].X() - fx) * (st[m].X() - fx) + (st[m].Y() - fy) * (st[m].Y()
- fy));
        if (df > tf) { df = tf;  f = m; }
        tt = sqrt((st[m].X() - tx) * (st[m].X() - tx) + (st[m].Y() - ty) * (st[m].Y()
- ty));
```

```
        if (dt > tt) { dt = tt;  t = m; }
    }
    if (f == t) return 0;   //起点和终点相同，不用乘车
    return MatPathCalc::tra.miniDist(f, t, d, r);
}
```