《数据结构与算法设计》
课程设计总结

学号：**2052082**

姓名：王辉

专业：计算机科学与技术

2022 年 9 月

目　录

第二部分 综合应用设计说明

2.1 题目

题号 2 ★★★上海的地铁交通网路已基本成型，建成的地铁线十多条，站点上百个，现需建立 一个换乘指南打印系统，通过输入起点站和终点站，打印出地铁换乘指南，指南内容包括起点站、换乘站、终点站。

（1）图形化显示地铁网络结构，能动态添加地铁线路和地铁站点。

（2）根据输入起点站和终点站，显示地铁换乘指南。

（3）通过图形界面显示乘车路径。

2.2 软件功能

2.2.1 查看地铁网络信息

1.图形化显示地铁网络；

2.能够查看网络中各线路的相关信息；

3.能够查看网络中各站点的相关信息；

4.能够放大缩小视图，方便查看地铁网络的细节；

2.2.2 给出换乘指南

1.能够选择出发路线、出发站、目标路线、目标站，选择获取三种策略的换乘指南；

2.给出距离最短的换乘指南，并在视图上区别显示出该乘坐路线；

3.给出换乘最少的换乘指南，并在视图上标注出换乘路线；

4.给出乘坐站点最少的换乘指南，并在视图上区别显示出该乘坐路线；

2.2.3 动态添加

1.动态添加地铁站点；

2.动态添加地铁线路；

3.动态添加站点的连接关系；

4.将添加的信息融入到原有信息中，在视图上显示，并参与换乘指南的决定。

2.2.4 其它

1.给出软件使用帮助；

2.给出制作者联系方式等信息。

实现方法在设计思想中阐述。

2.3 设计思想

1.实现思路

首先后端采用的数据结构是图，物理结构是图的邻接表结构。换乘策略的获取采用图的 djstl 算法。

其次前端的主要难点是图形化显示。有了第一题的设计实现经验后，对于 Qt 框架有了一定的认识了解。地铁网络的图形化显示可以采用 Qt 的视图—场景—图元三层框架。视图可以接受界面上的鼠标事件并提供缩放等功能；场景可以容纳各种图元控件；而地铁网络的

站点和线路可以用 Qt 的 QGraphicsEllipseItem 和 QGraphicsLineItem 图元控件进行绘制显示；地铁站点名的标注可以用 Qt 的 QGraphicsTextItem 进行关联显示；地铁站点和线路的详细信息可以利用图元控件的 setToolTip(QString tips)方法实现——当鼠标悬停于图元空间上时，即可显示 tips 字符串。至于乘坐路线，只需要将不在乘坐线路上的图元增加一定的透明度即可清晰的区别显示出来。整个地铁网络的所有图元最终就根据其经纬度坐标，容纳于一个场景之中，可以拖动场景使得其的不同部分显示在视图区域进行查看。

最后动态添加的功能，只需要将用户添加的站点、线路、连接关系等信息添加到原有的地铁网络信息中，再整体 update 一次，即可完成添加功能。这一部分的难点主要在如何尽可能限制用户的输入情况，避免错误输入引起程序崩溃。可以尽量使用下拉框，选择框等输入方式达到这一目的。

2.选用数据结构的理由

首先分析问题。地铁网络显然是一个图，严格来说是一个无向网，即站点间的边是有权值且没有方向的。因此选用的逻辑结构是图，站点是图的顶点，站点间的连接是图的边。软件核心功能——给出最短距离路线、最少换乘指南、最少站点路线，完全可以在图结构的基础上，使用图的 djstl 算法实现。

在图的多种存储结构之中，最终采用的是图的邻接表结构。首先，地铁网络中，站点数较多，多达三百多个，边相对较少，如果使用邻接矩阵进行存储，那么这一定是一个稀疏矩阵，存在较多的空间浪费，因此不宜使用邻接矩阵存储；其次，其它更为巧妙的数据结构，如十字链表等，实现较为复杂，不便于频繁操作，且学生对此不够熟悉，因此不宜使用这些结构。最后，学生对图的邻接表结构较为熟悉，且在第一题已经使用过图的邻接表的存储方式，故选用邻接表存储方式。

3.数据结构的设计思想

首先，为了便于访问，邻接表并不采用链表实现，而是用 Vector 容器代替单链表，便于随机访问。其次考虑到站点和线路的名字均唯一，可以将所有的站点和线路均存储于 Vector 容器中，用 Hash 表将站点名或线路名的字符串与其在 Vector 容器中的索引进行关联。因此在后端操作上，只需要使用站点或线路的名称即可，需要其具体信息时可以用 Hash 表获取其在对应 Vector 容器中的索引，即可快速访问。这样可以大大增加软件的性能。

4.算法设计的基本流程

本题使用到的核心算法是普通的 djstl 算法，算法的具体实现就不再赘述了。

2.4 逻辑结构与物理结构

前端：

逻辑结构：无向网

存储实现：

站点：

```
class Station
{
private:
    int id;                     // 站点标识 ID
    QString name;               // 站点名
    double longitude, latitude; // 站点经,纬度
    QSet<int>linesSet;          // 站点所属地铁线路的集合
    // 用静态属性记录所有站点的边界位置，便于绘制地铁线路图
    static double minLongitude, minLatitude, maxLongitude, maxLatitude;
```

```
    int distance(Station another);  // 求取站点 another 与本站点的距离
public:
    Station();  // 构造函数
    Station(QString name, double lng, double lat, QList<int>linesList); // 构造
函数
    // 声明友元便于获取私有属性
friend class SubwayGraph;
};
```

线路：

```
class Line{
private:
    int id;                     // 线路标识 id
    QString name;               // 线路名称
    QColor color;               // 线路颜色
    QSet<int>stationSet;        // 线路站点集合
    QSet<Edge>edges;            // 线路连接关系
public:
    // 构造函数
    Line(){ };
    // 构造函数
    Line(int id_, QString name_, QColor color_)
        :id(id_), name(name_), color(color_){};

    // 用声明友元替代一系列 get...()函数，方便获取私有属性
    friend class SubwayGraph;
};
```

地铁网络图

```
// 图的节点
class Node{
public:
    int id;         // 邻接点 id
    double distance;// 两点之间的距离
    // 构造函数
    Node(){};
    Node(int id_, double distance_):id(id_), distance(distance_){};
    // ">"运算符重载，用于构建小顶堆
    bool operator > (const Node& n)const { return distance > n.distance; }
};


class SubwayGraph
{
```

```
private:
    QString name;                           // 地铁名
    QVector<Station>stations;               // 地铁网络中所有的站点
    QVector<Line>lines;                     // 地铁网络中所有的线路
    // 用两个 hash 表提高访问地铁线和站点的效率
    QHash<QString,int>stationHash;          // 站点名到 stations 中存储位置的映
射
    QHash<QString,int>lineHash;             // 线路名到 lines 中存储位置的映射
    QSet<Edge>edges;                        // 所有连接关系的集合
    QVector<QVector<Node>>graph;            // 地铁网络图结构，近似于邻接表
    void makeGraph();  // 生成图的邻接表结构
public:
    // 构造函数
    SubwayGraph(){};
    SubwayGraph(QString name_):name(name_){ };
    // 获取线路名 Key 在 lineHash 中的 value
    int getLineIndex(QString lineName);
    // 获取站点名 Key 在 stationHash 中的 value
    int getStationIndex(QString stationName);
    // 从文件中读取格式化的数据
    bool readData(QString fileName);
    // 更新所有站点的边界经纬度坐标
    void updatePos(double lng, double lat);
    // 往 edges 插入一条边，不考虑翻转
    bool insertEdge(int first, int last);
    // 清除数据
    void clearData();
    // 求时间最短线路
    bool transferMinTime(int start, int end, QList<int>&stationsList,
QList<Edge>&edgesList);
    // 求换乘最少路线
    bool transferMinTransfer(int start, int end, QList<int>&stationsList,
QList<Edge>&edgesList);
    // 求最少站点路线
    bool transferMinStation(int start, int end, QList<int>&stationsList,
QList<Edge>&edgesList);
    // 根据站点索引列表的站点名列表
    QList<QString> getStations(QList<int>stationsIndexList);
    // 根据站点名获取其所属线路名的列表
    QList<QString> getLines(QString stationName);
    // 获取所有线路名的列表
    QList<QString> getAllLines();
    // 根据站点索引获取站点名
    QString getStationName(int i);
```

```
    // 根据边 Edge 获取线路
    QList<int> getLinesOfEdge(Edge edge);
    // 获取边 Edge 的颜色（正片叠底）
    QColor getEdgeColor(QList<int> linesIndex);
    // 获取经纬度坐标边界信息
    double minLongitude();
    double minLatitude();
    double maxLongitude();
    double maxLatitude();
    // 获取站点经度
    double getLongitude(int i);
    // 获取站点纬度
    double getLatitude(int i);
    // 根据线路索引获取线路名
    QString getLineName(int i);
    // 获取所有边的列表
    QList<Edge> getEdges();
    // 获取所有站点索引的列表
    QList<int> getAllStationsIndex();
    // 获取所有站点名的列表
    QList<QString> getAllStationsName();
    // 根据线路名获取该线路中所有站点名的列表
    QList<QString> getStationsOfLine(QString lineName);
    // 获取站点的线路索引列表
    QList<int>getLinesIndex(int i);
    // 插入一条线路
    bool pushLine(QString lineName, QColor color);
    // 插入一个站点
    bool pushStation(QSet<int>lineSet,QString name,double longi,double lati);
    // 插入一条边
    bool pushEdge(QList<int>linesIndex,Edge edge);
};
```

前端：

主要类：

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private:
    Ui::MainWindow *ui;
    SubwayGraph* mySubway;   // 后端网络对象
```

```
    QLabel* labLongiLati;   // 状态栏标签 1——经纬度
    QLabel* labViewCord;   // 状态栏标签 2——View 坐标
    QLabel* labSceneCord;  // 状态栏标签 3——Scene 坐标
    QLabel* labMaker;  // 状态栏标签 4——制作者
    QGraphicsScene* Scene;  // 场景对象
    // 画一条边
    void drawEdge(Edge edge, bool flag = 0);
    // 画边的集合
    void drawEdges(QList<Edge>edges1, bool flag = 0, QList<Edge>edges2 = {});
    // 画一个站点
    void drawStation(int i, bool falg = 0, bool flag2 = 0);
    // 画站点的集合
    void drawStations(QList<int>stations1, bool flag = 0, QList<int>stations2 =
{});
    // 用经纬度计算对应场景坐标
    QPointF calScenePos(double longi, double lati);
    // 用场景坐标计算经纬度
    QPointF calLongiLati(QPointF scenePos);
    // 获得线路集的字符串形式
    QString getLinesName(QList<int>linesIndex);
    // 更新起始路线和终点路线的下拉列表
    void updateLinesItem();
    // 更新起始站的下拉列表
    void updateStartStationsItem();
    // 更新终点站的下拉列表
    void updateEndStationsItem();
    // 初始化视图
    void initGraphics();
    // 画视图（将路径与网络区别显示）
    void drawGraphics(QList<int>stationsindex,QList<Edge>edges);
private slots:
    void on_mouseMovePoint(QPoint point);  // 鼠标移动
    void on_confirmBtn_clicked();  // 确认按钮
    void on_startLine_currentIndexChanged(const QString &arg1);  // 开始路线下
拉列表当前内容改变
    void on_endLine_currentIndexChanged(const QString &arg1);  // 终点路线下拉
列表当前内容改变
    void on_exitAction_triggered();   // 退出程序按钮
    void on_addAction_triggered();   // 动态添加按钮
    void on_zoomoutAction_triggered();  // 放大按钮
    void on_zoominAction_triggered();   // 缩小按钮
    void on_recoverAction_triggered();  // 恢复网络视图按钮
    void on_aboutMakerAction_triggered();  // 关于制作者页面按钮
    void on_helpAction_triggered();   // 帮助页面按钮
```

```
signals:
    void zoomout();   // 放大视图
    void zoomin();   // 缩小视图
};
```

2.5 开发平台

开发平台	
计算机型号	联想小新 Air-14IIL 2020
计算机内存	16.0GB
CPU	Intel Core i7-1065G7 1.50 GHz
操作系统	Windows 10 家庭中文版
开发语言	C++（C++11 标准以上）
开发框架	Qt
集成开发环境	Qt 5.14.2 MinGw 32-bits
编译器	MinGW 7.3.0 32-bit for C++
编辑器	Qt Creater 4.11.1

运行环境：可在上述开发平台环境下运行，或者可以在打包文件中运行。

2.6 系统的运行结果分析说明

2.6.1 调试与开发过程

2.6.1.1 调试方法

与第一题的调试方法相同，虽然是第一次使用 Qt 框架以及 Qt 框架提供的编辑器 Qt Creater，但是 Qt 的调试与使用较多的 VS2019 编辑器基本相似，其提供的调试工具和 VS 2019 一样方便实用。在本次开发中，主要使用的调试方法是结合原因分析、输出调试、设置断点调试运行、单步调试运行、添加监视等方法。除了输出调试使用与 C++的输入输出流不同以外，其它调试方法基本相同。在 Qt 框架中是调用 qDebug()<<str 将信息输出到控制台窗口进行调试。输出调试主要用于检查一些运行结果不合理的错误，设置断点和单步运行调试则主要为了检查一些指针越界等的严重错误。

2.6.1.2 开发过程

软件开发大致分为三个过程。

1.先实现后端类，用控制台输出检查纠正错误。

2.根据需要在 Qt Creater 提供的设计师界面编辑窗口中设计软件界面。

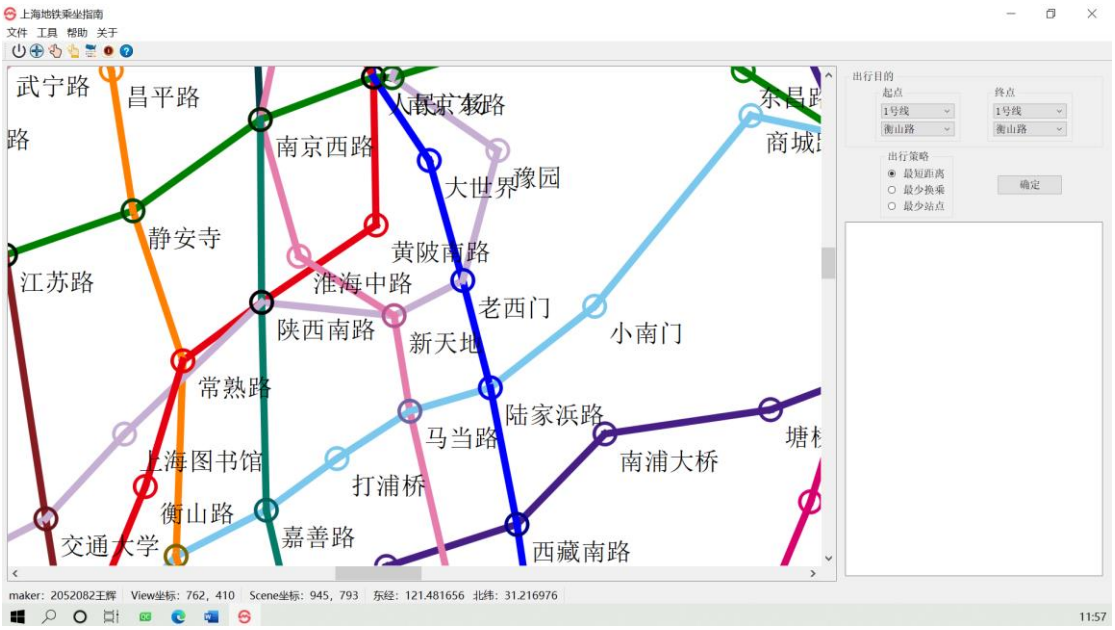
3.实现前端类，在软件界面上显示后端数据，图形化显示地铁网络，并接受用户输入，如鼠标移动等信号。

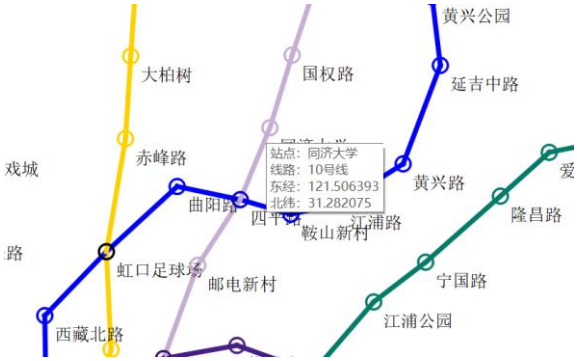
2.6.2 实现成果

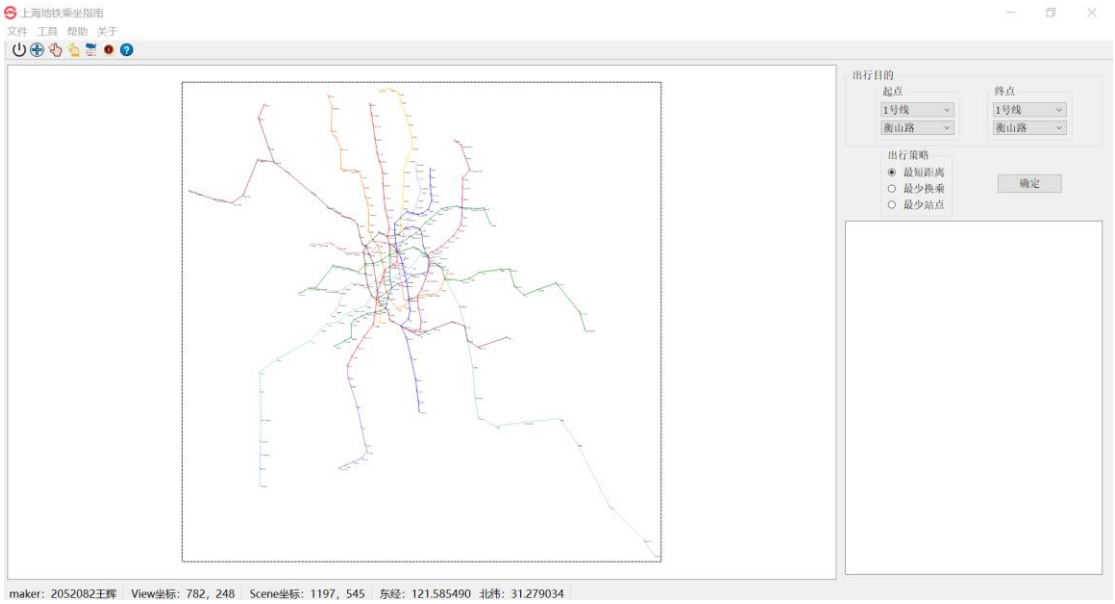
经过多次验证，程序能够按照预期运行。程序的正确性、稳定性和容错能力较高。

2.6.2.1 正确性

1.能够很好的图形化显示上海地铁网络。支持缩放查看。以及鼠标悬停在站点、线路上方时，显示详细信息。



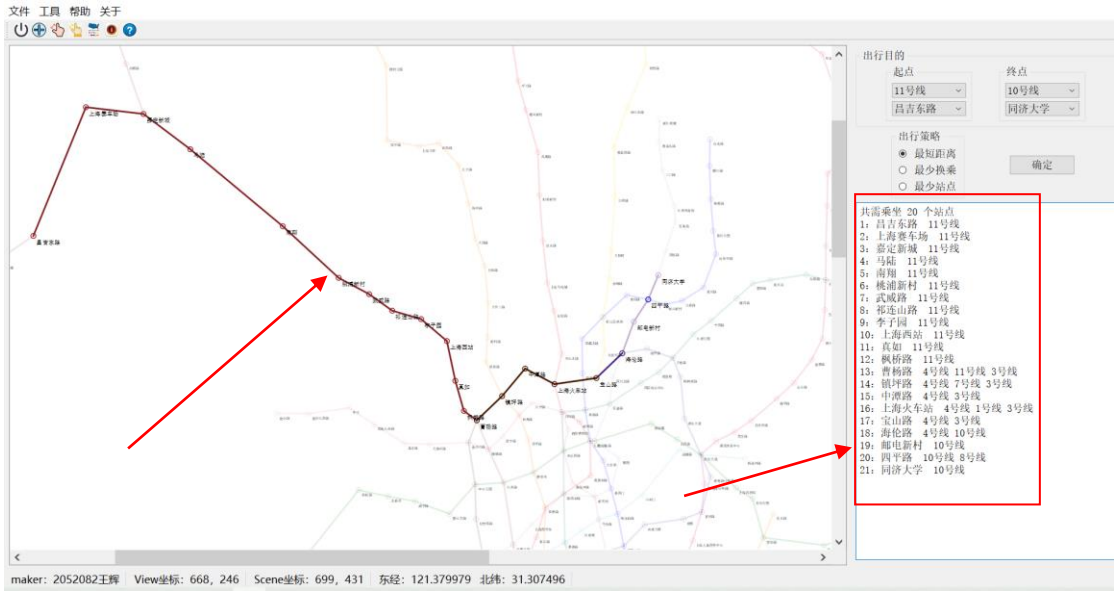




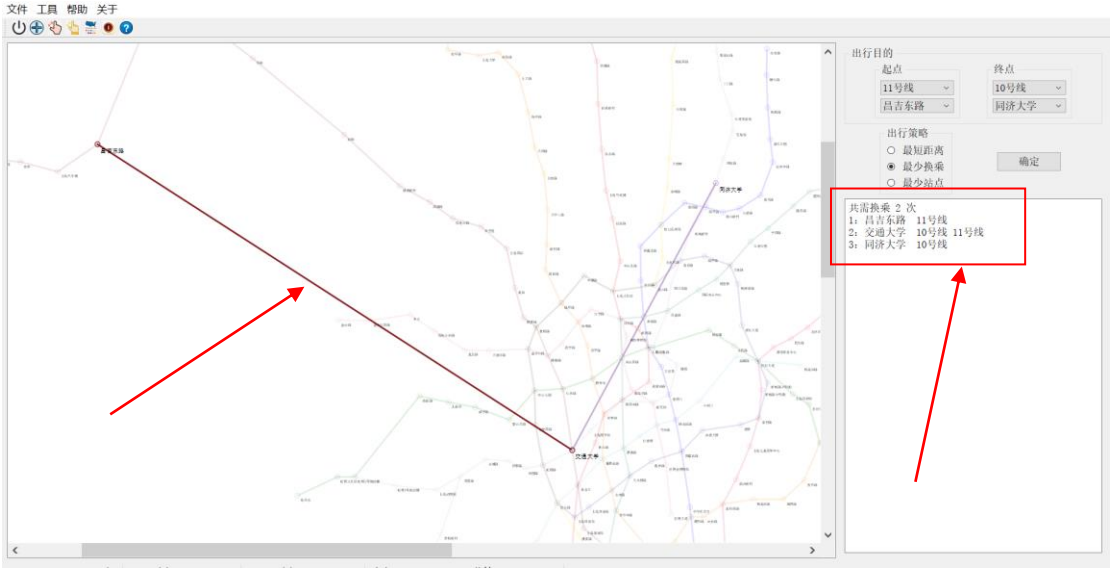
2.能够正确给出换乘指南。

如：从 11 号线昌吉东路到 10 号线同济大学

最短距离乘坐指南：

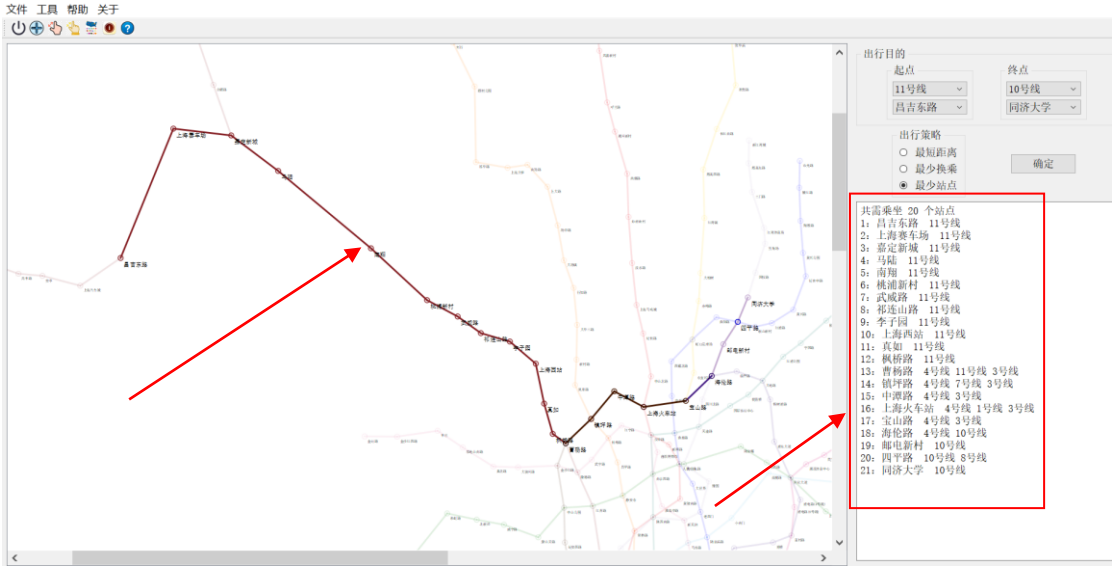


最少换乘乘坐指南：（只给出起始站、目标站、换乘站及其相应的地铁路线）



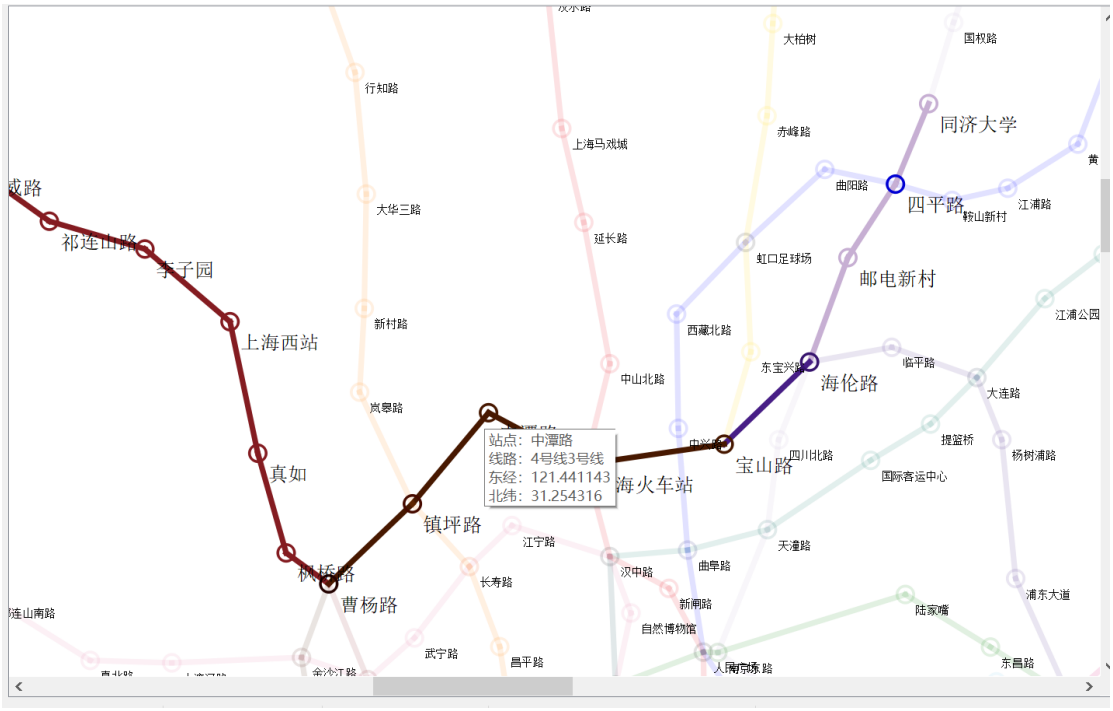
最少站点乘坐指南：

由于上海地铁设计合理，事实上，大部分最短距离乘坐路线与最好站点乘坐路线相同。



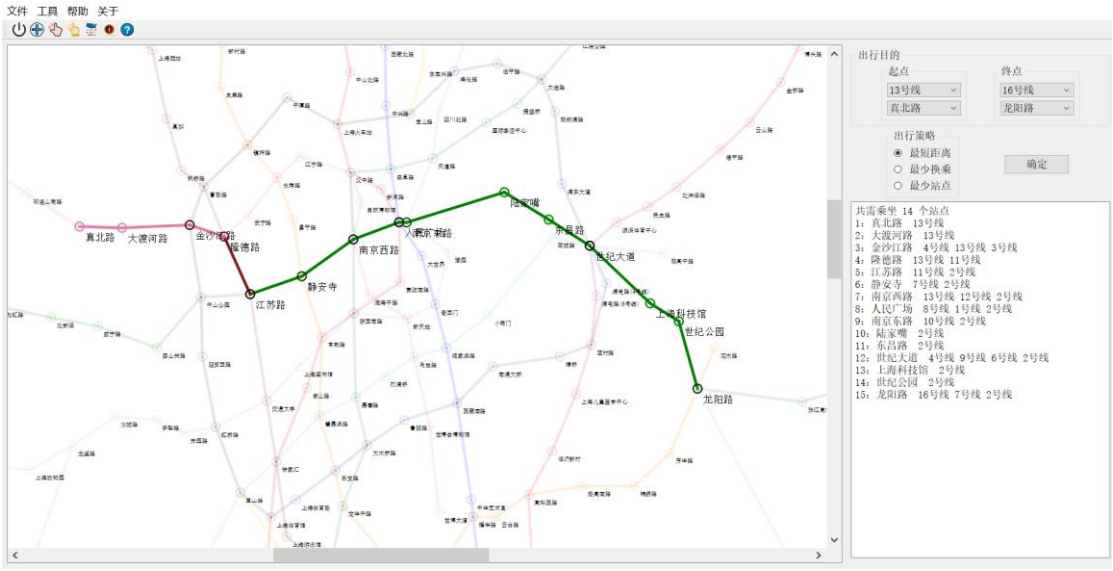
细节图：

共需乘坐 20 个站点
1：昌吉东路 11号线
2：上海赛车场 11号线
3：嘉定新城 11号线
4：马陆 11号线
5：南翔 11号线
6：桃浦新村 11号线
7：武威路 11号线
8：祁连山路 11号线
9：李子园 11号线
10：上海西站 11号线
11：真如 11号线
12：枫桥路 11号线
13：曹杨路 4号线 11号线 3号线
14：镇坪路 4号线 7号线 3号线
15：中潭路 4号线 3号线
16：上海火车站 4号线 1号线 3号线
17：宝山路 4号线 3号线
18：海伦路 4号线 10号线
19：邮电新村 10号线
20：四平路 10号线 8号线
21：同济大学 10号线

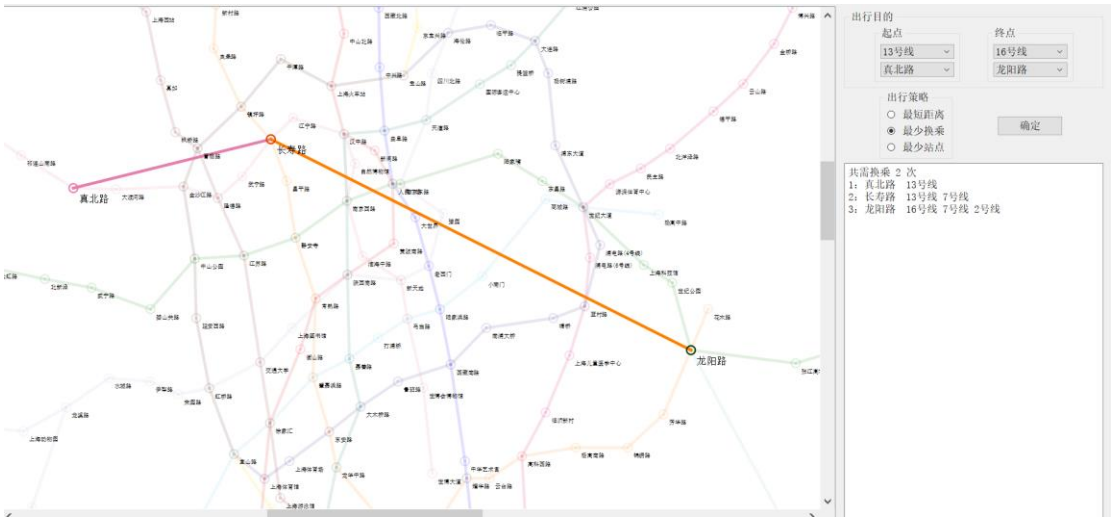


再比如：从 13 号线真北路到 16 号线龙阳路

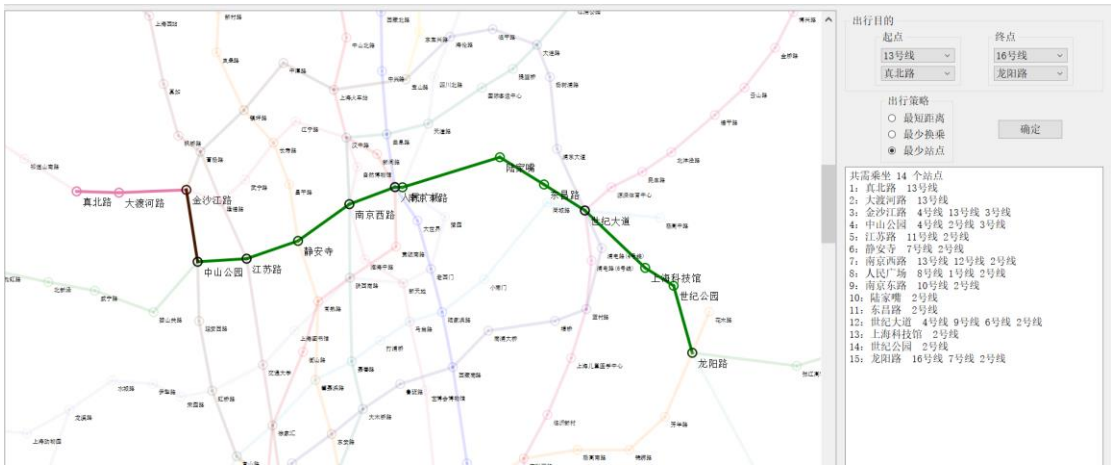
最短距离乘坐指南：



最少换乘乘坐指南：

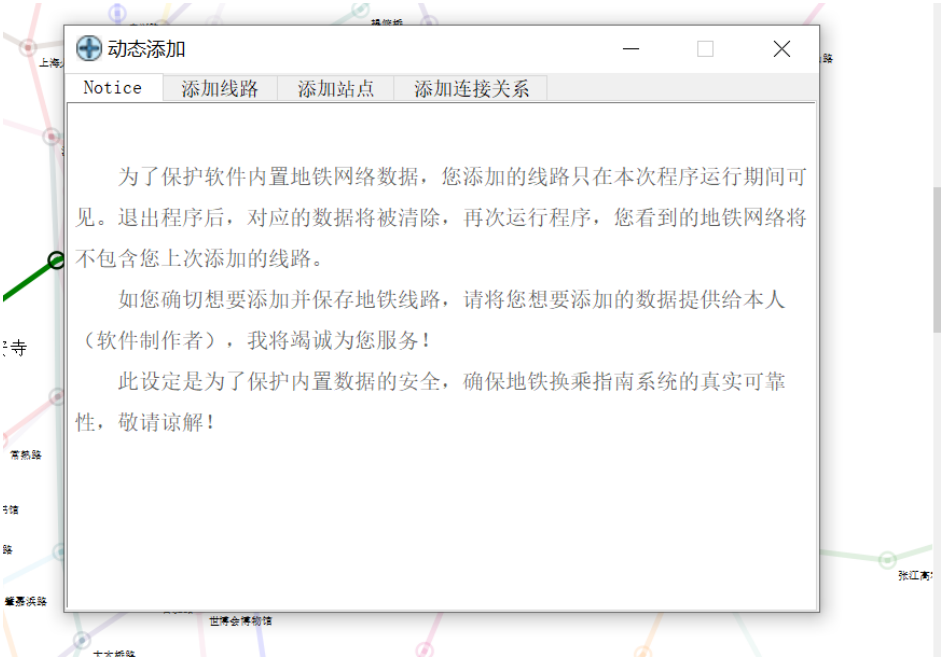


最少站点乘坐指南：

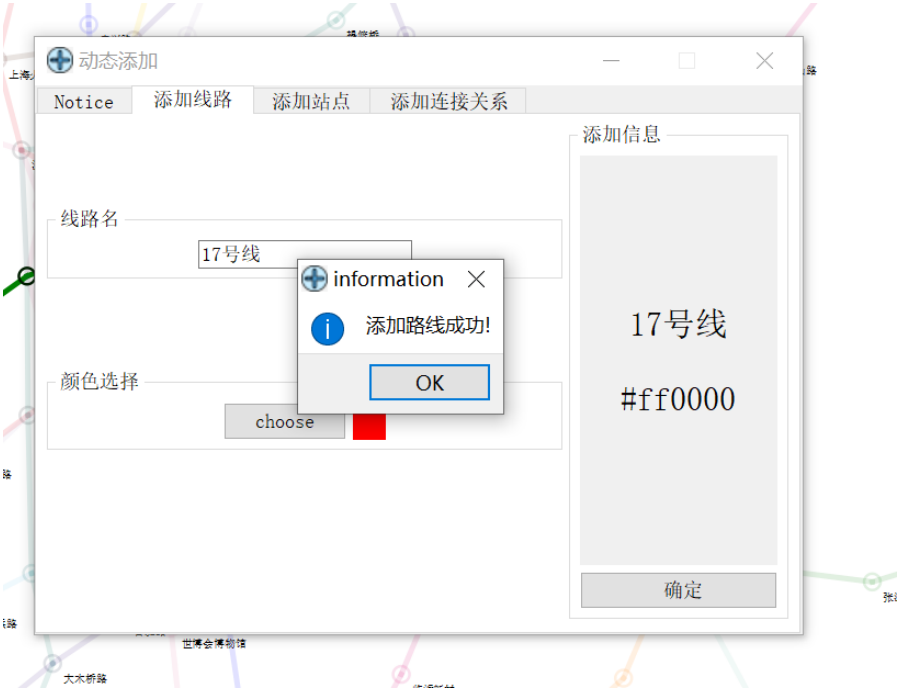


3.能够正确动态添加站点、线路、连接关系。

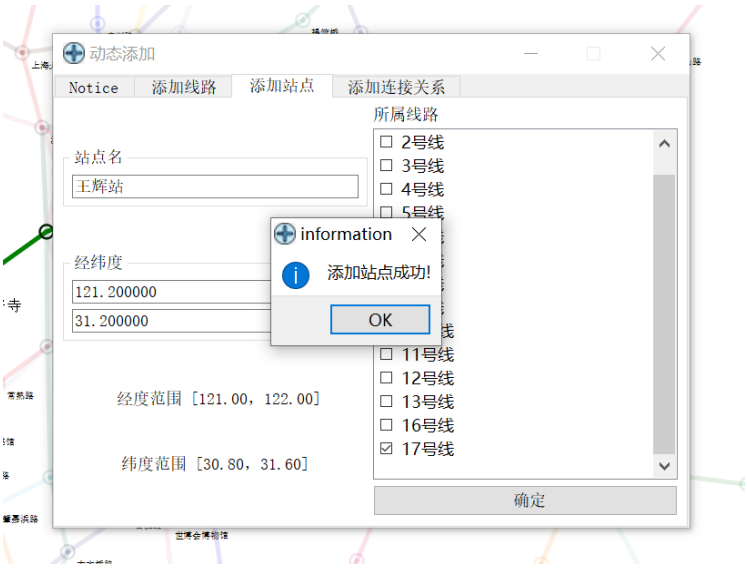
动态添加需知：

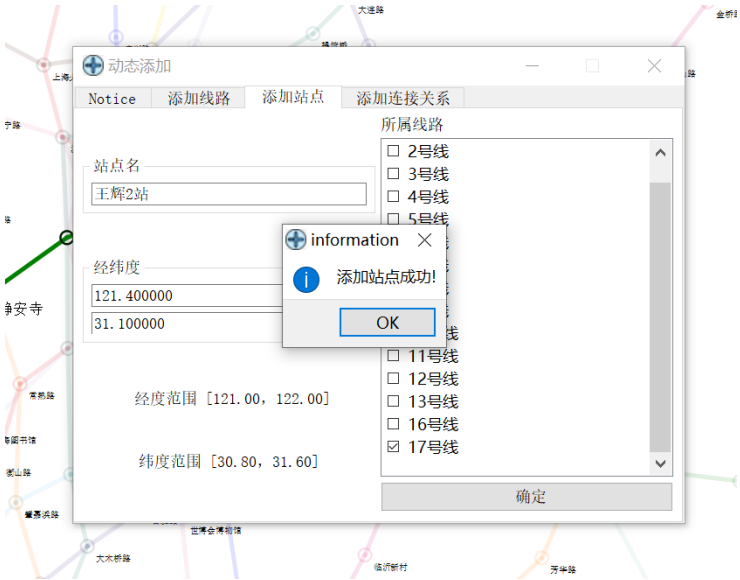

动态添加
Notice
添加线路
添加站点
添加连接关系
为了保护软件内置地铁网络数据，您添加的线路只在本次程序运行期间可见。退出程序后，对应的数据将被清除，再次运行程序，您看到的地铁网络将不包含您上次添加的线路。
如您确切想要添加并保存地铁线路，请将您想要添加的数据提供给本人（软件制作者），我将竭诚为您服务！
此设定是为了保护内置数据的安全，确保地铁换乘指南系统的真实可靠性，敬请谅解！

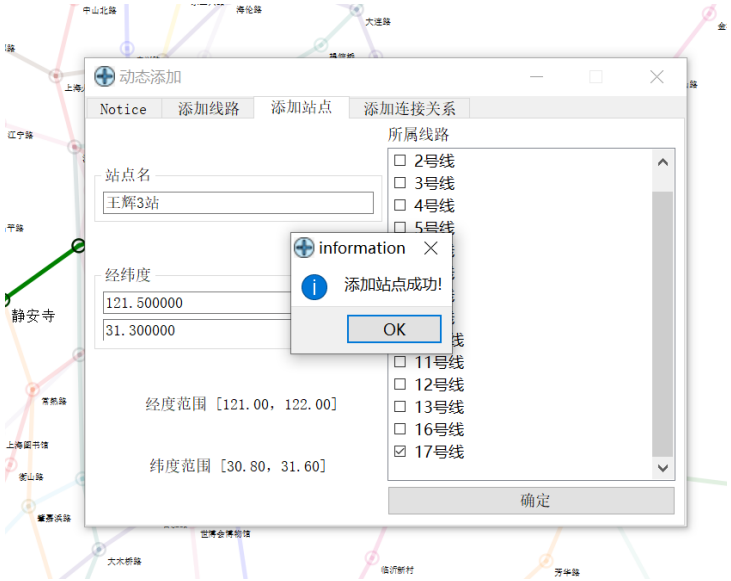
添加线路：


动态添加
Notice
添加线路
添加站点
添加连接关系
线路名
17号线
颜色选择
choose
information
添加路线成功!
OK
添加信息
17号线
#ff0000
确定

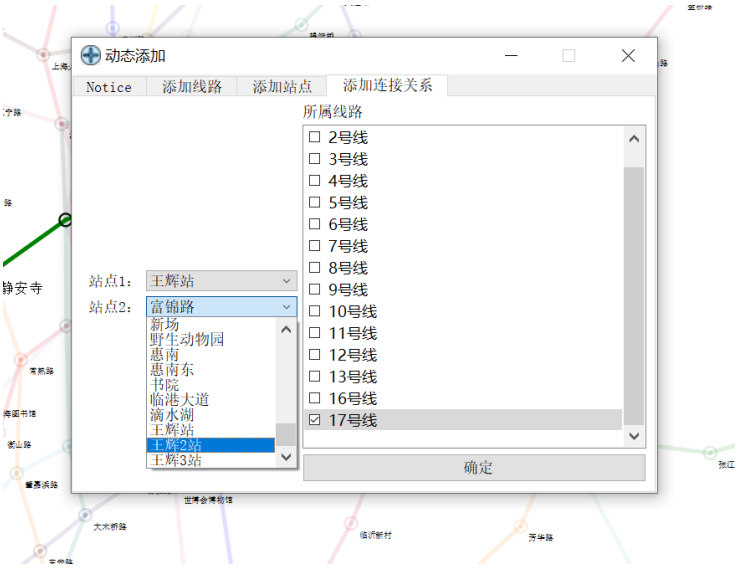
添加站点

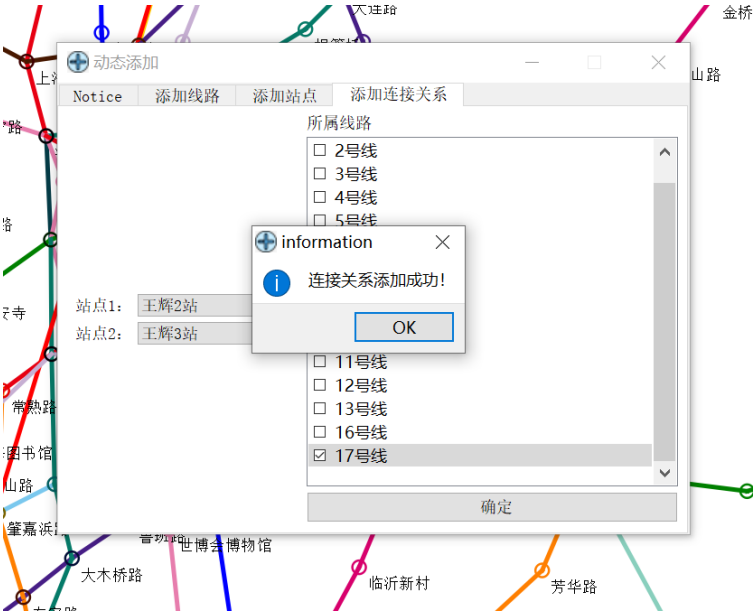

动态添加
Notice
添加线路
添加站点
添加连接关系
所属线路
站点名
王辉站
经纬度
121.200000
31.200000
information
添加站点成功!
OK
经度范围［121.00，122.00］
纬度范围［30.80，31.60］
2号线
3号线
4号线
11号线
12号线
13号线
16号线
17号线
确定

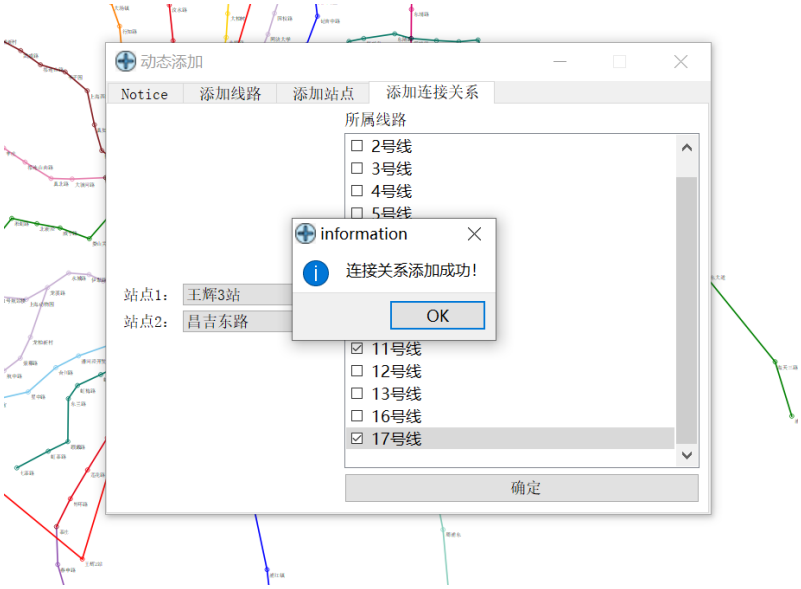

动态添加
Notice
添加线路
添加站点
添加连接关系
所属线路
站点名
王辉2站
经纬度
121.400000
31.100000
information
添加站点成功!
OK
经度范围［121.00，122.00］
纬度范围［30.80，31.60］
2号线
3号线
4号线
11号线
12号线
13号线
16号线
17号线
确定


动态添加
Notice
添加线路
添加站点
添加连接关系
所属线路
站点名
王辉3站
经纬度
121.500000
31.300000
information
添加站点成功!
OK
经度范围［121.00，122.00］
纬度范围［30.80，31.60］
2号线
3号线
4号线
11号线
12号线
13号线
16号线
17号线
确定

添加连接关系：

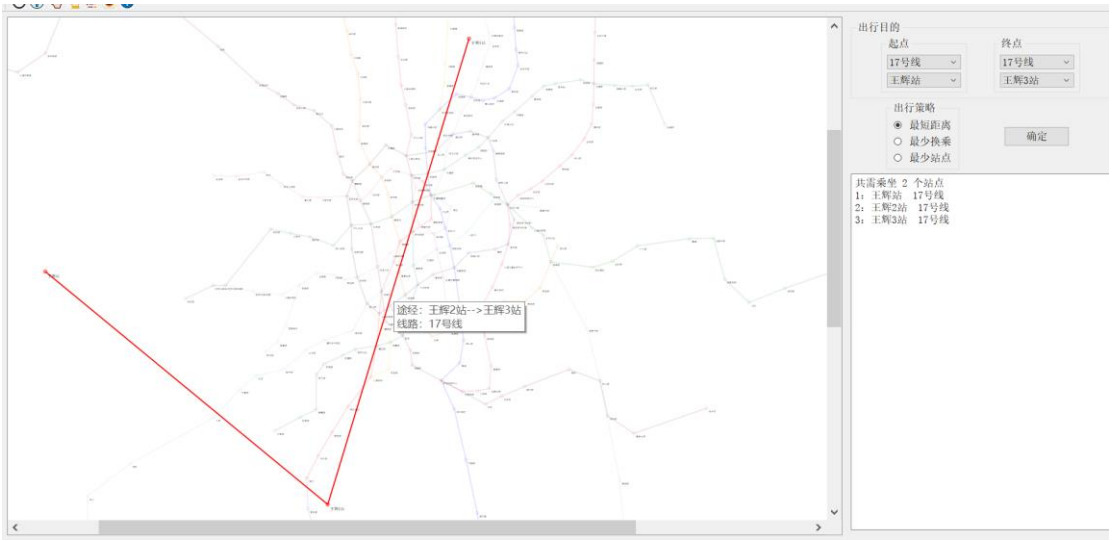

动态添加
Notice
添加线路
添加站点
添加连接关系
所属线路
2号线
3号线
4号线
5号线
6号线
7号线
8号线
9号线
10号线
11号线
12号线
13号线
16号线
17号线
站点1：
王辉站
站点2：
富锦路
新场
野生动物园
惠南
惠南东
书院
临港大道
滴水湖
王辉站
王辉2站
王辉3站
确定

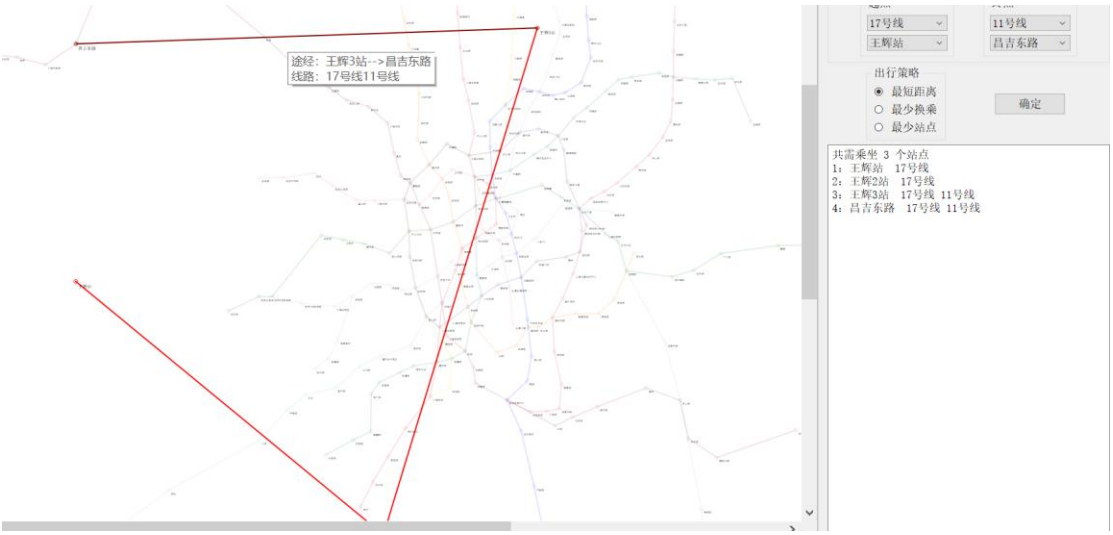

动态添加
Notice
添加线路
添加站点
添加连接关系
所属线路
2号线
3号线
4号线
11号线
12号线
13号线
16号线
17号线
information
连接关系添加成功!
OK
站点1：
王辉2站
站点2：
王辉3站
确定
金桥
大木桥路
临沂新村
芳华路

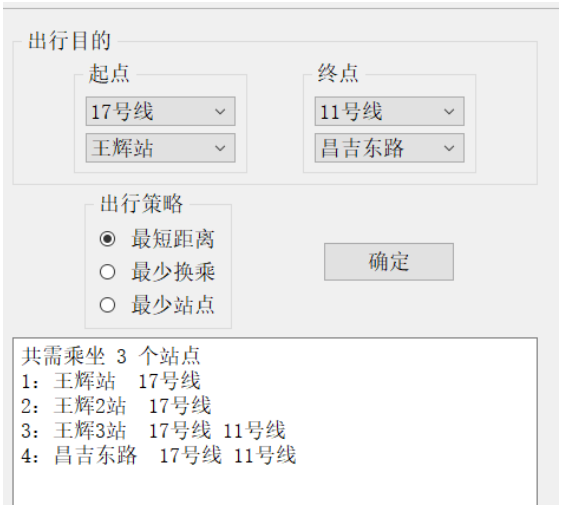

动态添加
Notice
添加线路
添加站点
添加连接关系
所属线路
2号线
3号线
4号线
11号线
12号线
13号线
16号线
17号线
information
连接关系添加成功!
OK
站点1：
王辉3站
站点2：
昌吉东路
确定

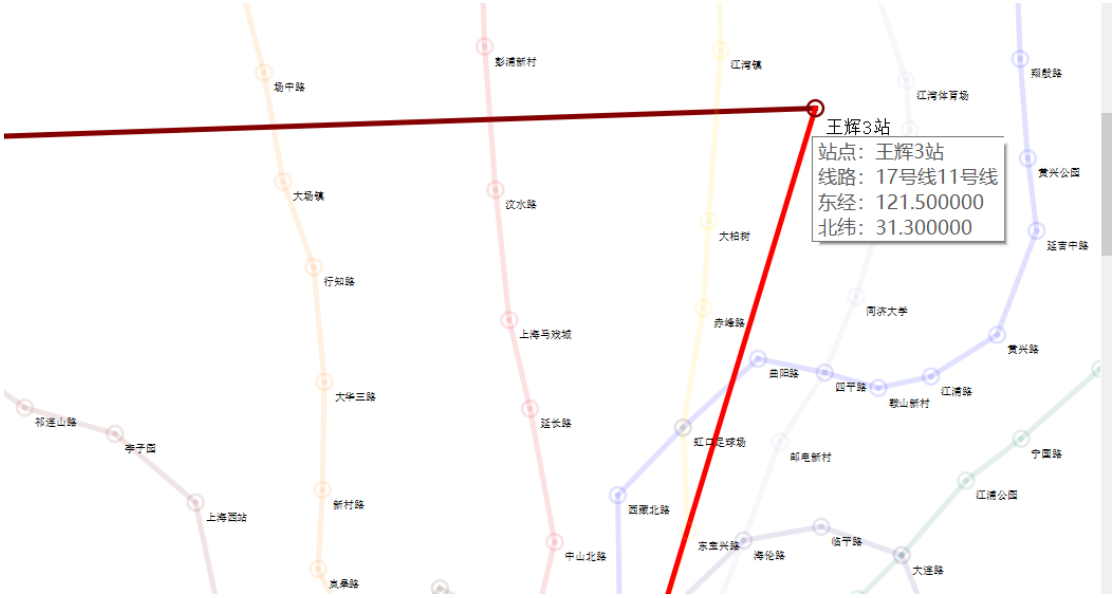
添加结果：

三个站点均为胡乱添加，但是可以看到地图上已经成功显示了添加的站点和线路，在查询换乘策略时，也可以选择自己添加的站点和线路，并正确给出换乘指南。



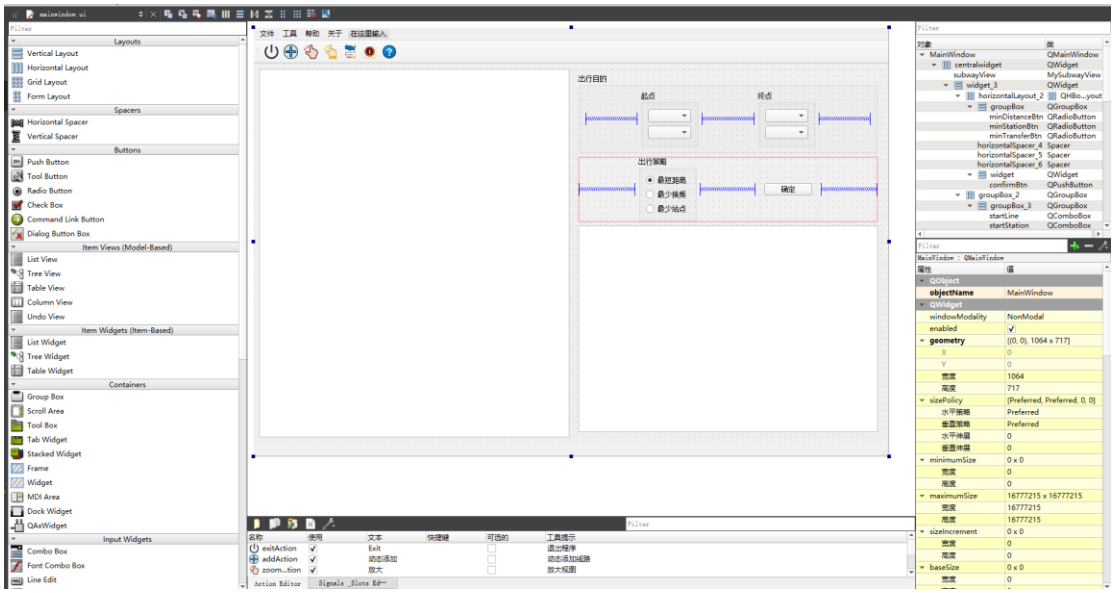


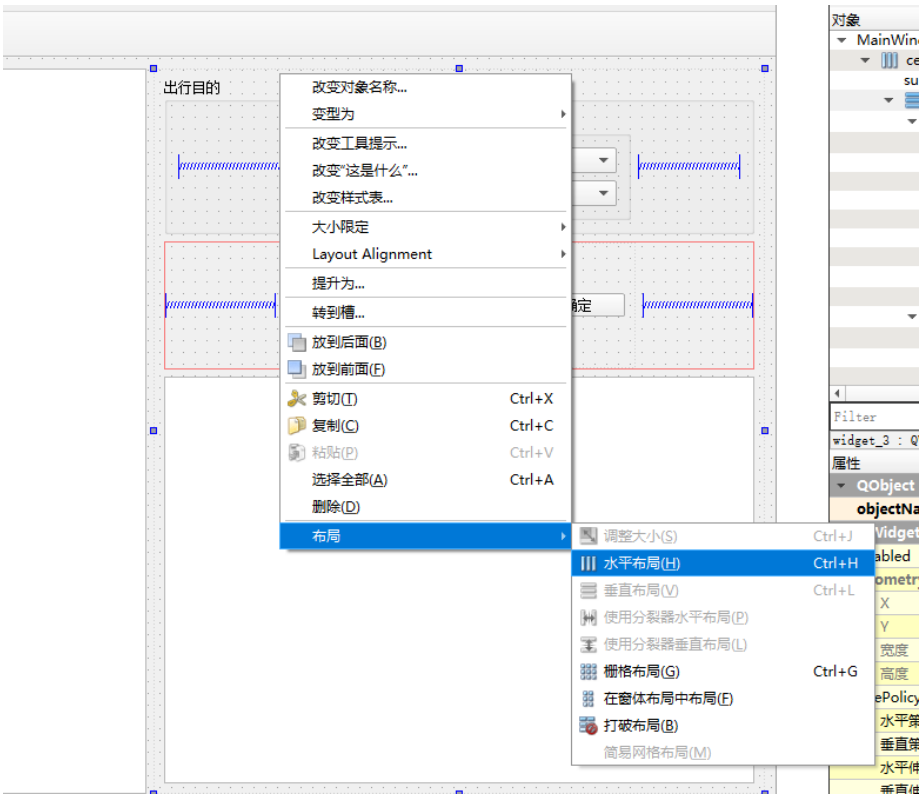




2.6.2.2 稳定性

1. 界面稳定性：使用 Qt 提供的界面布局方法，以及在 Qt 的设计师编辑界面随时调整软件界面各窗口的大小等属性，使得软件界面有很好的稳定性。





2. 视图的缩放功能、拖动功能都能稳定执行。

3. 鼠标悬停显示详细信息的功能偶尔得不到响应，或者响应较慢，这一点是不足的地方，原因依然值得分析和探讨。有可能是因为场景中的图元控件太多，彼此之间相互交叠，视图检测到鼠标事件后，不知该响应哪一图元控件的信号。

4. 视图内容、下拉列表等都能够得到快速的更新。比如换乘指南基本上不需要时间就能得到，非常快速稳定；比如在线路下拉选择框选择了路线之后，在站点下拉选择框就能立刻更新显示该路线的所有站点以供选择；比如在动态添加连接关系后，视图立刻就能把该线路显示在地铁网络中……

5. 经过多次反复验证，除了第三点不足外，软件具有较高的稳定性。

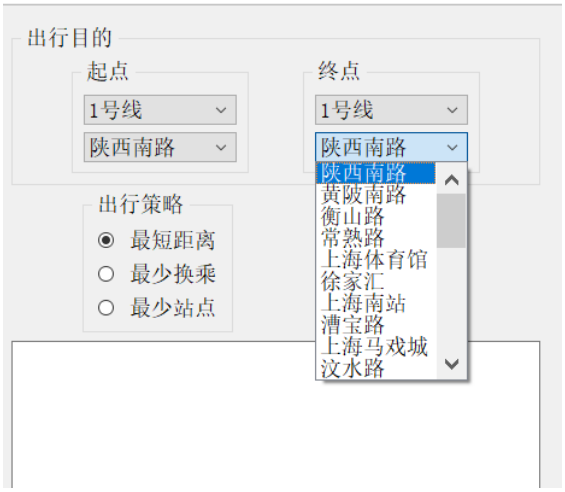
2.6.2.3 容错能力

从多角度避免错误，并给出充分的对话框信息提示，以提高程序的容错能力。经过多次反复实验验证，程序的容错能力较强。

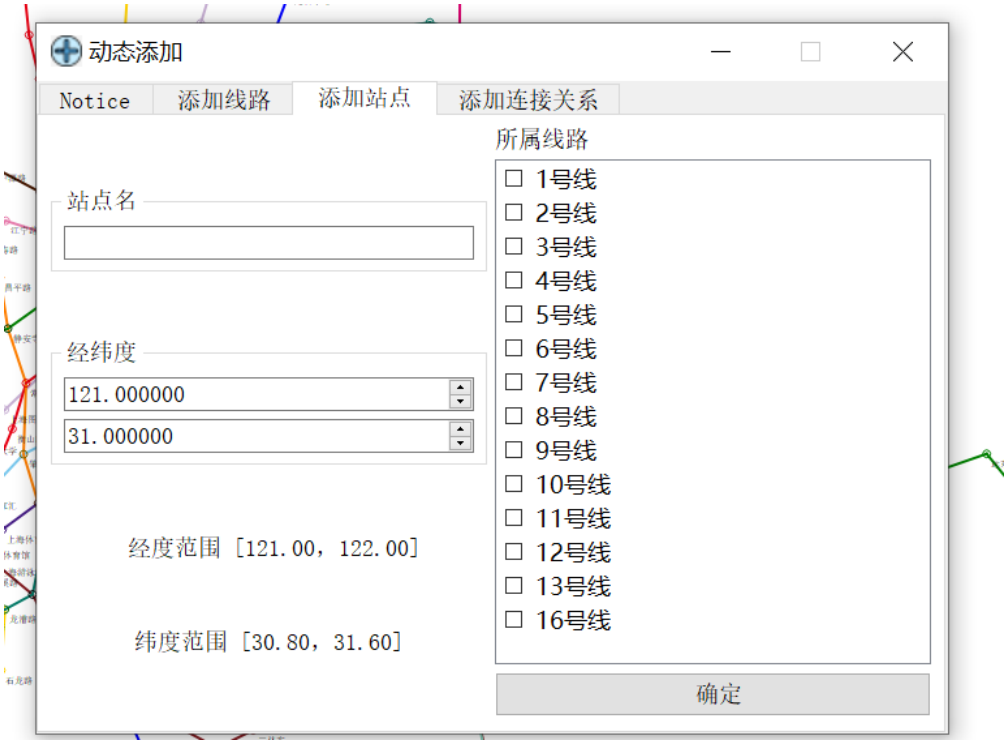
限制输入

1 多使用下拉选择框以及固定格式的输入框。

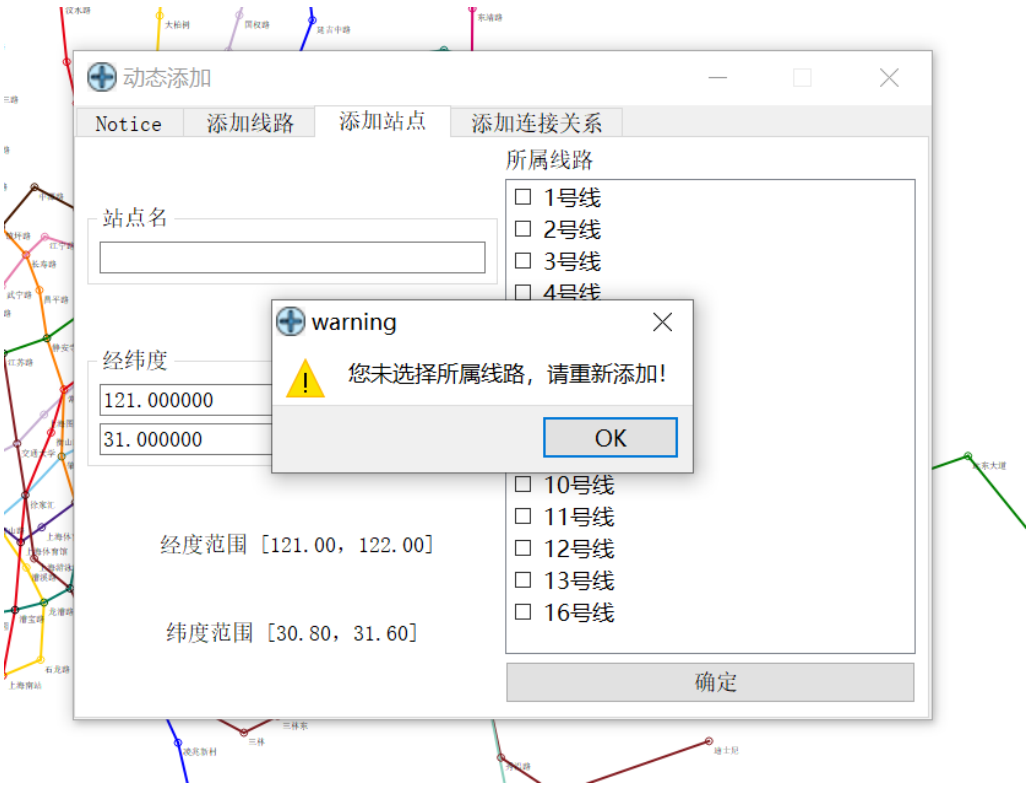
从右图可以看到，用户只能出下拉框中选择线路站点，只能出单选按钮中选择出行策略，因此此处不可能出现非法操作。考虑充分的前提下，几乎不会有错误发生。

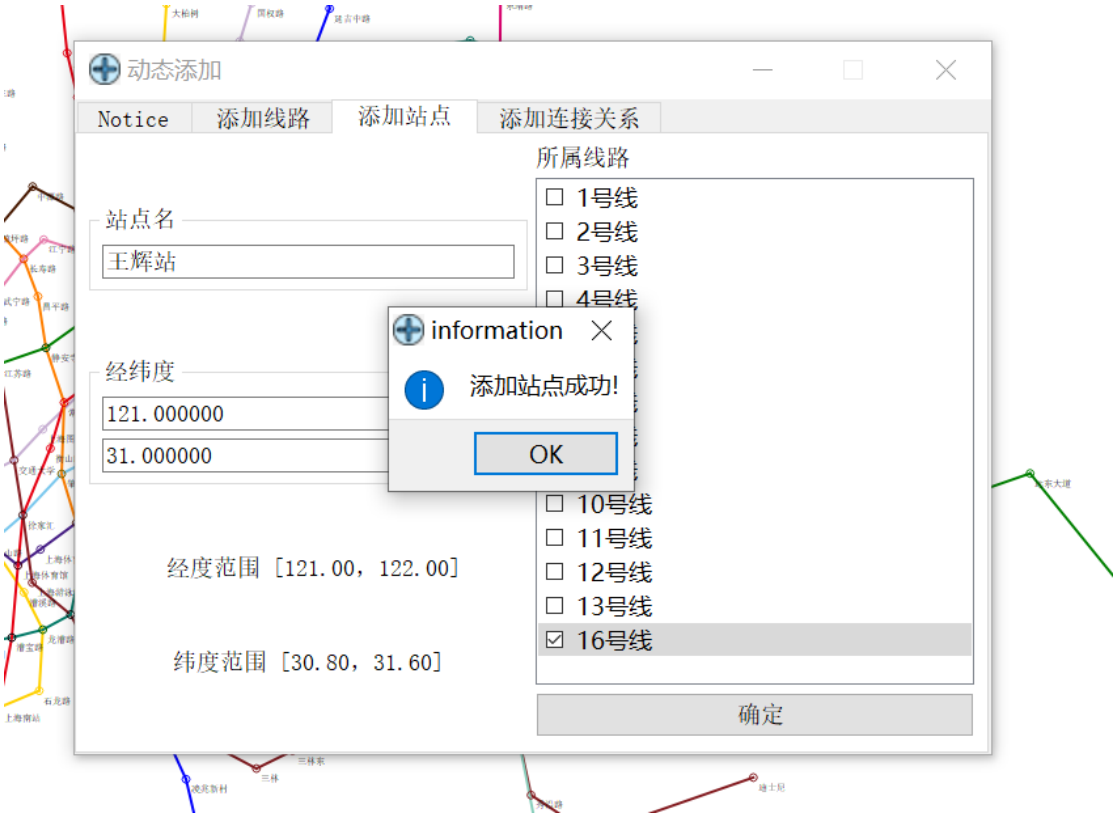


如下如所示，经纬度的输入使用了固定格式的输入框，并有着输入范围限制。对于线路也只需要从复选框中进行选择。因此在添加站点时，用户只需要文本输入站点名即可，可以避免很多错误。还有很多类似的处理，都是为了限制输入的情况，避免非法输入。



充分的对话框提示





2.7 操作说明

2.7.1 获取换乘指南

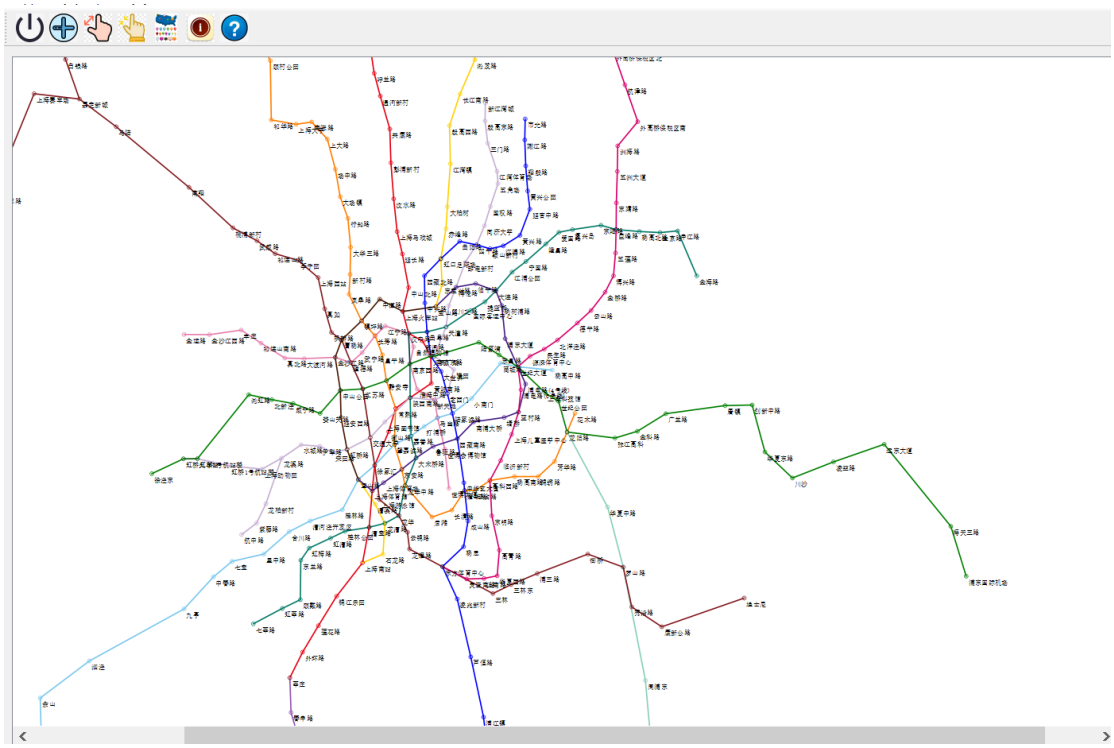
操作：下拉框选择起始线路 → 下拉框选择起始站点 → 下拉框选择目标线路 → 下拉框选择目标站点 → 单选框选择出行策略 → 点击确定，即可获得对应的换乘路线。

2.7.2 查看视图

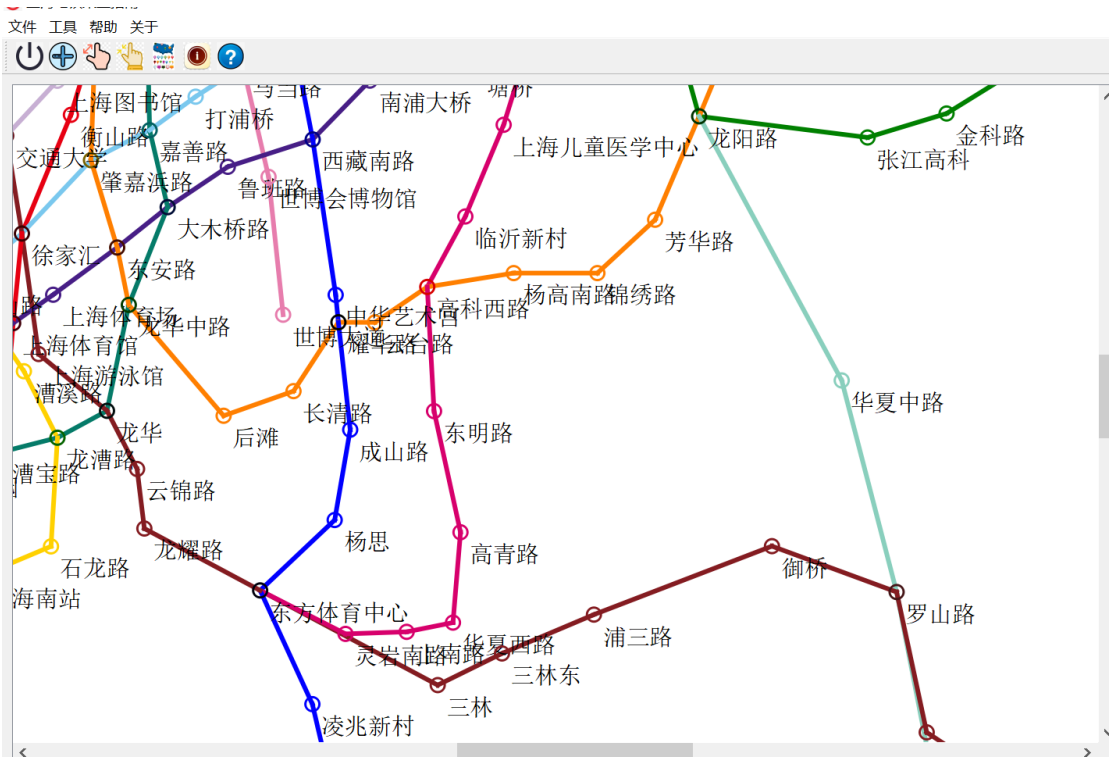
1.鼠标右键按下后移动，或移动下侧或右侧滑条，可以拖动以显示视图的不同区域；

2.鼠标位于视图区域时滚动滑轮可以以鼠标位置为中心缩放视图；

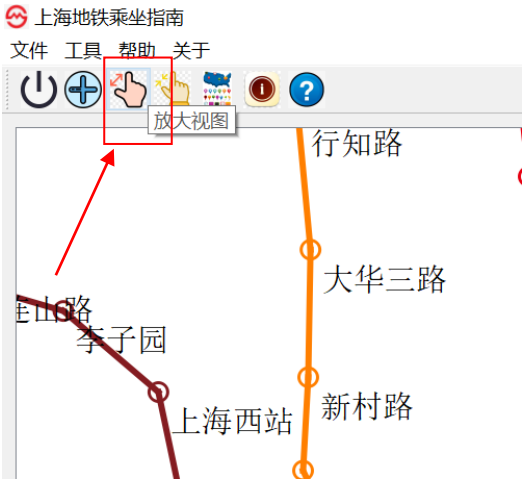
下图：缩小视图效果。

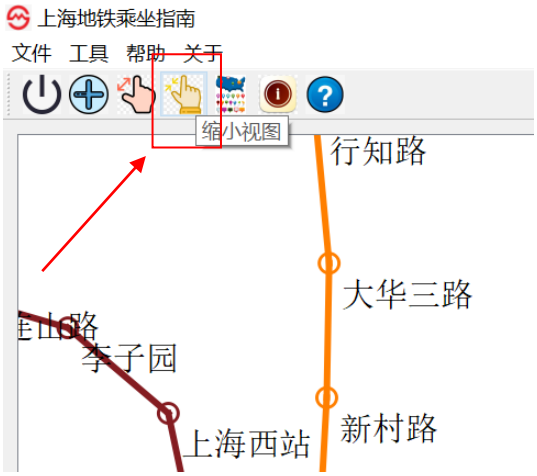


下图：放大视图效果



3.点击工具“放大”也可放大视图，点击工具“缩小”也可缩小视图；





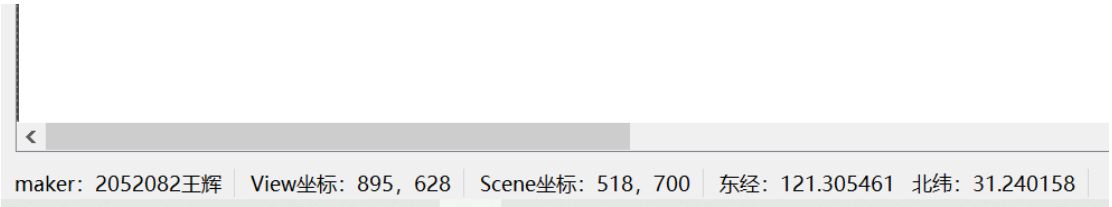
4.点击工具“恢复视图”，可把地铁网络图恢复成不透明状态。

如下图：大部分图元透明

点击“恢复视图”工具后变成全部不透明，如下图：

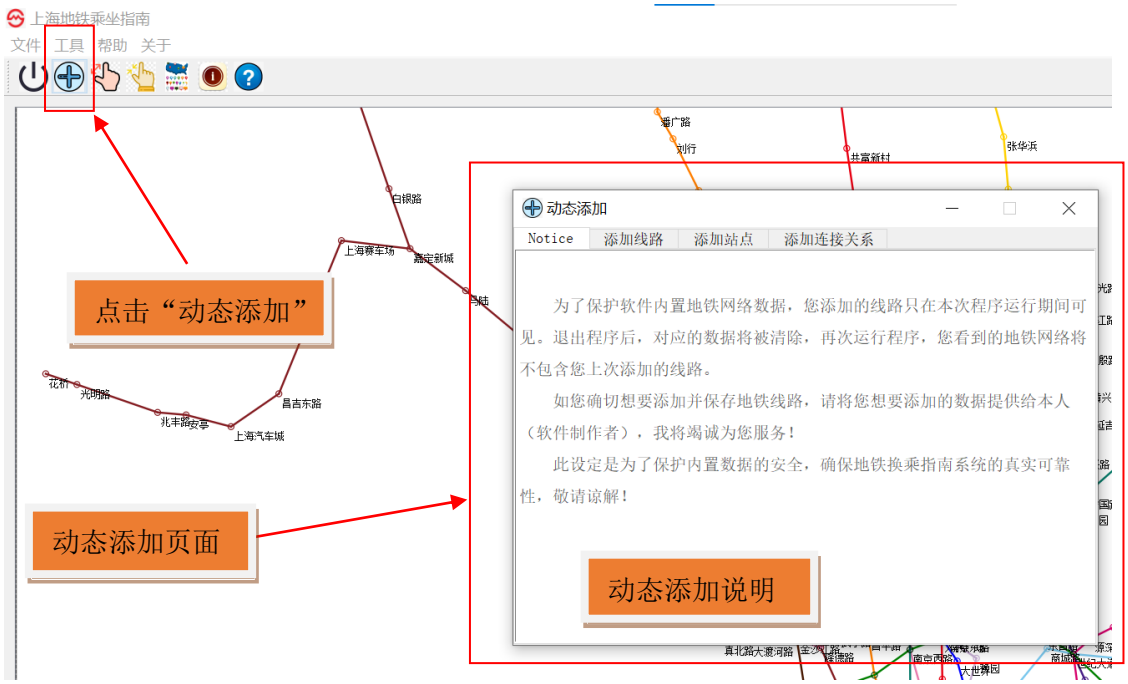
5.鼠标悬停于站点图元上，可显示站点详细信息；鼠标悬停于线路图元上，可显示线路信息。（截图无法体现鼠标悬停的效果）

6.状态栏可查看鼠标所在位置对应的经纬度坐标。



2.7.3 动态添加

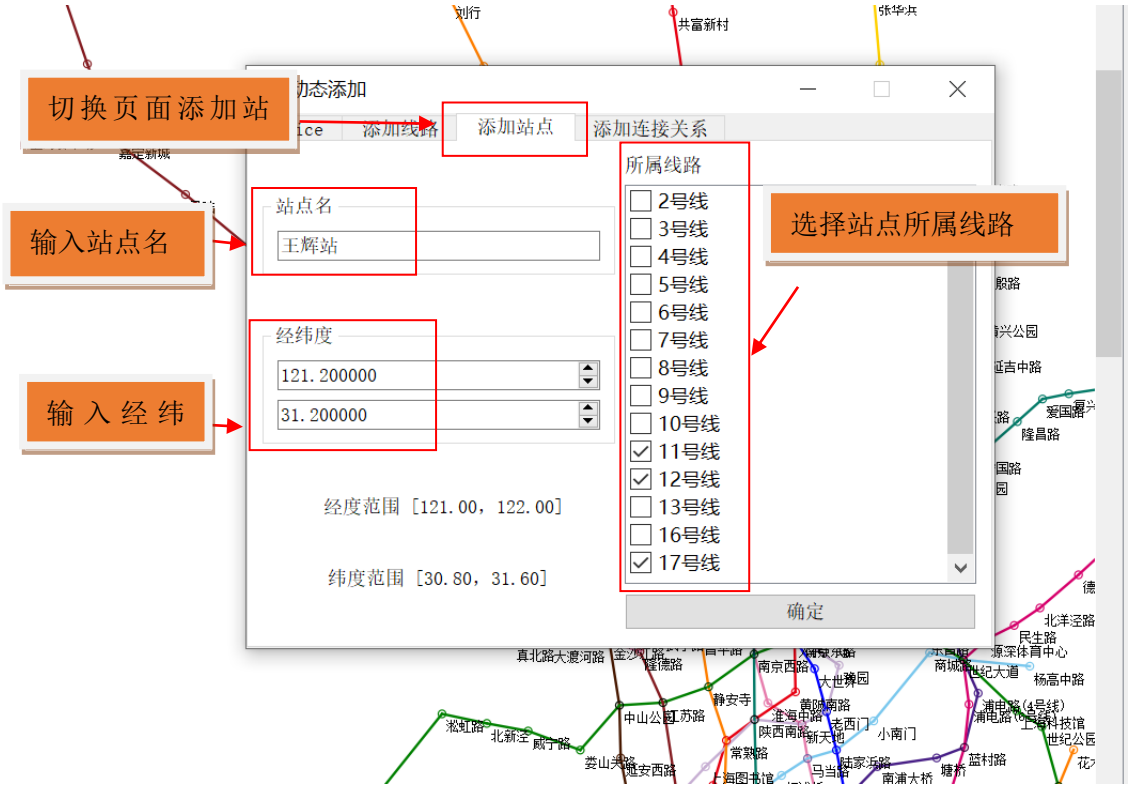
1.点击工具“动态添加”可弹出动态添加页面



2.添加线路

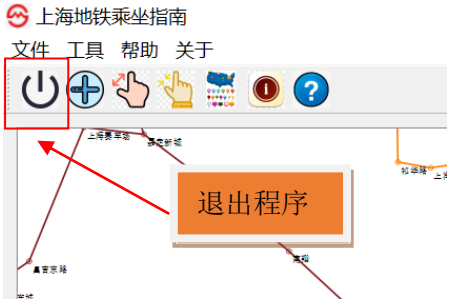

动态添加
Notice
添加线路
添加站点
切换页面添加线路
输入线路名
线路名
17号线
颜色选择
choose
选择线路的颜色
添加信息
17号线
#55aa00
显示已添加的信息
确定

3.添加站点


切换页面添加站
输入站点名
输入经纬
动态添加
添加线路
添加站点
添加连接关系
站点名
王辉站
经纬度
121.200000
31.200000
经度范围 [121.00, 122.00]
纬度范围 [30.80, 31.60]
所属线路
选择站点所属线路
2号线
3号线
4号线
5号线
6号线
7号线
8号线
9号线
10号线
11号线
12号线
13号线
16号线
17号线
确定

4.添加连接关系（添加连接关系后可在地铁网络视图中看到新添加的线路）

2.7.4 使用帮助、关于、退出程序



第三部分 实践总结

3.1.所做的工作

1.系统的学习了 Qt 的基本框架；学习了 Qt Creater 编辑器的使用，尤其是在设计师界面进行软件界面编辑。

2.重点学习了 Qt 的 QGraphicsView-QGraphicsScene-QGraphicsItem 平面绘图框架。

3.完成了算法实现题的全部工作，包括设计数据结构，代码实现，软件设计、测试等。

4.完成了综合应用题的全部工作，包括设计数据结构，代码实现，软件设计、测试等。

5.完成了一份 52 页的万字实验报告。

3.2.总结与收获

1．学习掌握了很多知识。

首先，对于数据结构理论课程中所学的知识，经过一个学期，已经遗忘了很多。经过这次作业，我又重新捡起了数据结构书籍，尤其重点复习了图的知识。

其次，对 C++语言的掌握更加扎实。在过去，为了应付作业，基本上都是使用面向过程的编程方式，对于 C++三大特性的利用其实很少。而在本次项目中，从头到尾使用面向对象的思想，这迫使自己再次去学习 C++特性的利用，尤其对于封装，继承，重载等操作达到了一定的熟悉程度。

最后，我学会了使用 Qt 可视化框架，学会了 Qt 的信号槽机制，学会了 Qt 的平面绘图框架，学会了 Qt Creater 的设计师编辑界面的软件界面编辑操作……

2.获得了独立开发大型复杂项目的实践经验。

在平时的课程学习中，往往没有独立开发项目的机会，稍有工作量的大作业也都是以小组形式完成，因此每次都只是参与其中的一小部分。而且由于自身的惰性以及时间有限，每次都会挑选自己熟悉的后端模块去完成，因此对于项目开发的全部过程，尤其是前端界面和前后端连接工作并没有太多的接触和经验。而本次项目开发是个人工作，而且有很长的假期，提供了充分的时间保障，本人可以充分的进行前期知识的学习准备，细致的进行项目方案设计，并独立地逐一完成各个模块。可以说，经过这次项目开发，我的代码能力有了较大的提升，真正掌握了一个实用的可视化框架——Qt 框架。这比之以往只会在控制台输出是一个很大的进步。

3.一些感悟

在正式实现项目前，我总是想做好充分的知识准备，典型的行为就是在 B 站上看完了一整套漫长的 Qt 视频教程。但是我发现，真正开始项目的时候，之前学习的很多东西用不上，或者需要用却记不起来了，这时候有需要再次去学习。所以，有时候不必要追求做好所有的准备，或者说根本就很难做好。项目开发的过程中，一定会发现还是有很多问题，很多不懂，这时候就需要点对点，有针对性的去学习对应的知识。大概就是“缺什么我就补什么”，“兵来将挡没水来土掩”的道理。

开发项目时，一定要边开发边测试，开发一部分就测试一部分。有时因为懒惰，不愿意对逐个模块进行测试；当代码量大起来后，如果运行出错，调试就会变得更加困难。很有可能就会卡住很久，浪费大量的时间。边开发边测试，就能尽可能避免在模块合并之后出现很多问题，减小后期调试的工作量。

4.最后，感谢努力的自己！感谢老师！