上海地铁换乘系统设计说明

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

本程序为一个较为完备的地铁网络线路展示系统，功能主要包括以下几个方面：

1、图形化显示上海地铁网络结构，并支持放大、缩小等功能，可以显示某一站点的详细信息等等；

2、查询两站点之间的路线，支持最短路线查询和换乘最少路线查询两种模式，并将查询结果显示在文字界面和图形化界面上；

3、支持添加线路和站点，添加完成后可以进行图形化显示，并可以用于 1 和 2 的所有功能中

2.3 设计思想

2.3.1 总体设计思想

本题相较于上一题而言是一道综合性强、涵盖范围广、实用性强的题目。对于这种大型工程，不可能一次设计出完全适合的数据结构和算法。为此，我采用了敏捷开发的思想，结合在软件开发中学习到的 UML 建模思想，先从整个系统的功能需求大致推导出需要的各个类和数据结构，按照完整的功能链需求列出各个类之间的关系，快速开发出一个基础版本。然后，再对该版本逐步进行完善，得到更加完善的版本。

由于本题没有涉及到动画播放、延迟等等方面的内容，故算法和图形界面的代码可以实现完全分离。这对于面向对象设计是一件很好的事情。在代码结构的设计中，我充分利用了面向对象的开发思想，为每个可以抽象出来并且具有一些类似操作的部分都设计了相应的

类，如站点类、线路类、地铁系统类、图形界面管理类等等。各类之间的关系也非常明确，比如线路类中含有多个站点类成员，地铁系统类中含有多个线路类成员等等。

在开发过程中，首先大致设计出后端的各种类和数据结构，并且加以实现。然后再逐步实现前端的界面，过程中将后端操作与前端的按钮等进行连接，实现前后端相连。

2.3.2 各模块具体设计思想

本题的数据结构设计以及类的划分大多是按照实际情况来的。每个站点都有自己的信息，故我建立了一个 Station 类表示站点，包含名称、经纬度、所属线路等等相关信息。而站点又是通过线路进行组合的，故我建立了一个 Line 类表示线路，该类中存储了线路的名称、颜色、包含站点、总站点数等等信息。此外，整个地铁线路图是一个整体，故我建立了一个 SubwaySystem 类，用于表示一整个地铁系统，包含多个线路和站点的信息。这种类设计也体现了一种自顶向下、自下而上的思想。

在数据结构方面，由于站点、线路是名称和实体对象之间进行一一对应，故我在很多地方使用了 Qt 中的 QMap 这一数据结构。QMap 类似 c++中的 map，是基于红黑树实现的一种键值相对应的类型，使用效果非常类似于哈希表，但是其查找的复杂度为 O(logn)，稍慢于哈希表。程序中由于地铁站点和线路数目有限，故使用 QMap 也可以达到很高的效率。

在如地铁站点所属线路等方面，我还使用了集合这一数据结构。Qt 中的集合类是 QSet，集合的特性使得元素不能重复插入集合中，非常适合某些特殊场合的要求。

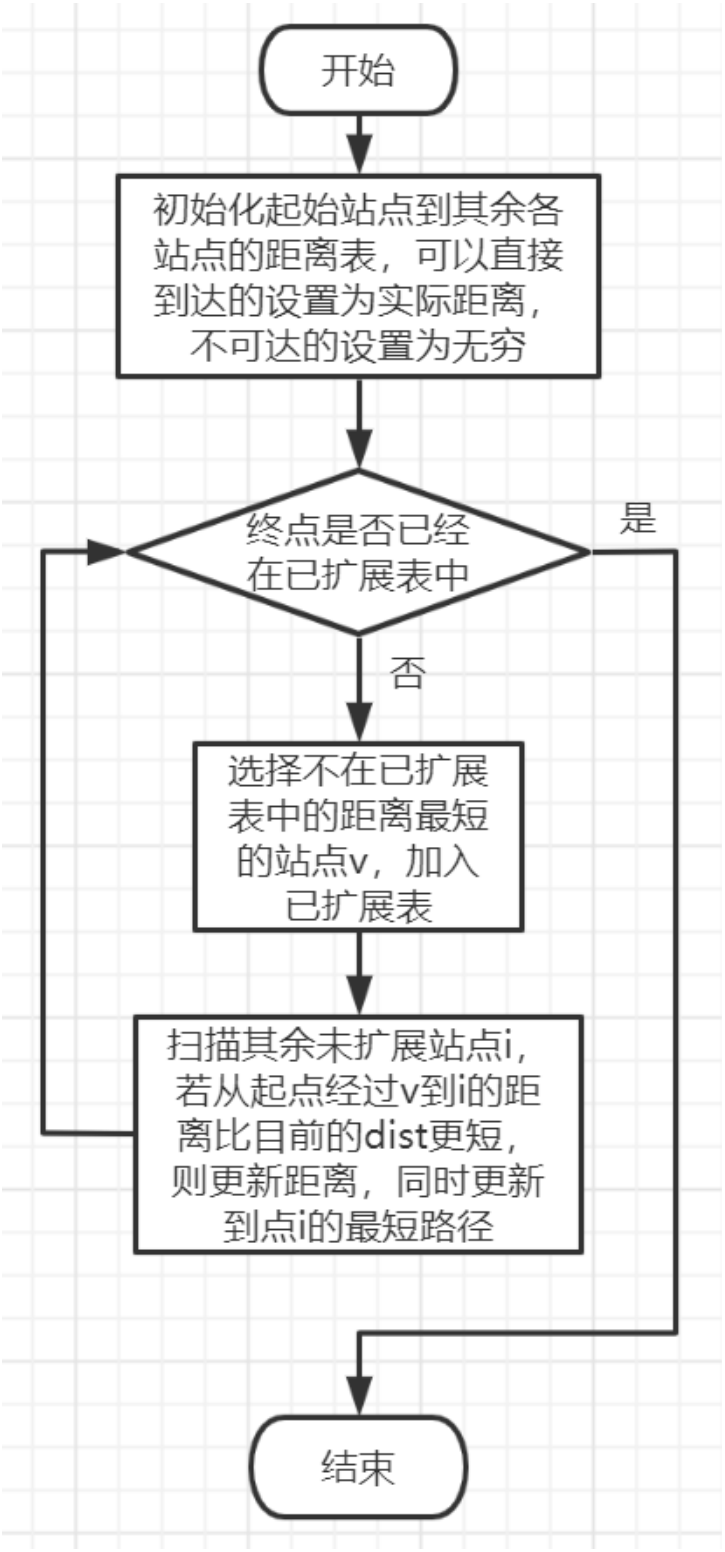
在本题的算法设计部分，最关键的部分是如何查询最短路径以及最少换乘路径。这两个问题我采用了两种不同的实现方法。

最短路径问题是典型的使用 Dijkstra 算法进行求解的问题。同时，通过查阅各种资料，我了解到还有一种 SPFA 算法，见**错误!未找到引用源。**和**错误!未找到引用源。**，也可以很好地求解单源最短路径问题。

SPFA 算法要对所有的边去进行一次松弛操作，进行了 n-1 次更新，先初始化距离数组，起点赋值为 0，其余赋值为无穷大，先起点入队列，入了队列的被标记，当队列不为空时循环，队首元素出队，松弛与队首元素相连的边，这些被更新的点如果不在队列中就加入队列，队首元素继续出队，松弛与队首元素相连的边，是不需要去找离原点最近的点的，所以 Dijkstra 算法用的是小根堆优化，SPFA 直接用的队列优化。

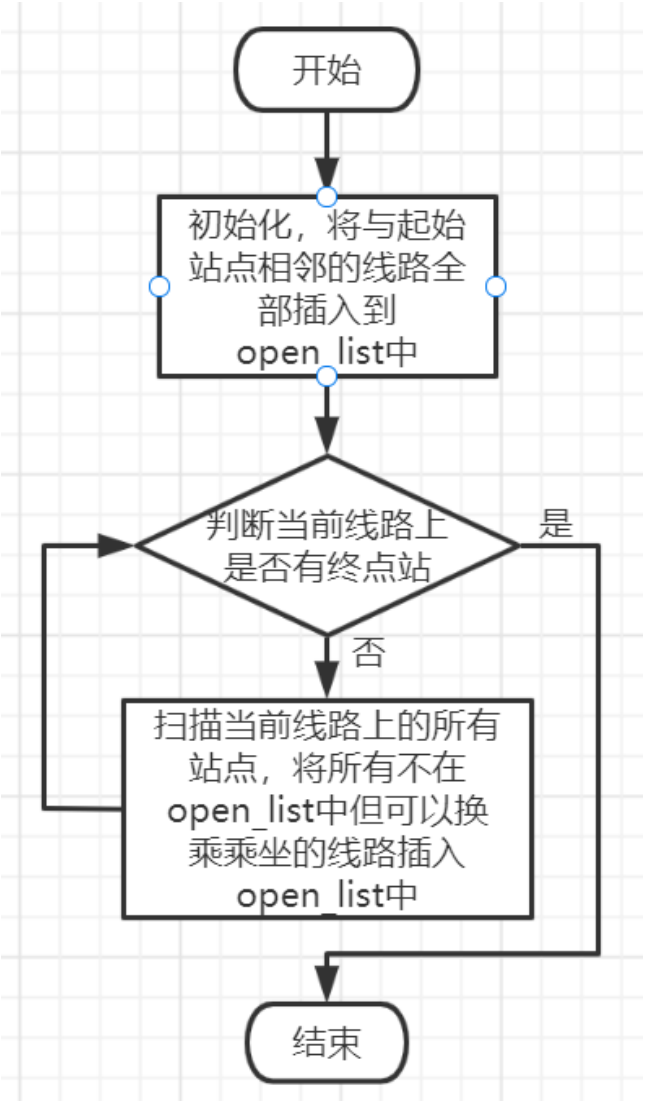
此外，SPFA 算法可以处理边权值为负的情况。由于地铁站点之间的距离一定是正数，

因此这一点利用不到。在程序中，我使用了 Dijkstra 求解最短路径问题。算法的主要流程如下（**为简化算法流程图，没有标出不存在任何路径的情况**）：



最少换乘路径问题无法使用上述算法进行求解，因为无法定义距离。如果以换乘的次数作为某站点的距离，那么将会出现某站点到另一站点换乘次数反而减少的情况，而且这种情

况是很不好判断的，需要记录前面经过的所有站点，因此也不适用 SPFA 算法。程序中，我使用了广度优先遍历思想进行了算法设计。从起始站点开始，依次遍历所有不需要换乘、需要换乘 1 次、需要换成 2 次，……的站点，直到找到了终点站。算法流程如下：



2.4 逻辑结构与物理结构

本题考察的知识点较为综合，故数据结构使用的也非常丰富。

逻辑结构方面，集合结构、线性结构、树形结构、图形结构均有涉及。集合结构用于记录站点所属线路等等不可重复、对顺序无要求的信息。线性结构的使用非常广泛，许多地方使用了 QVector、QList 等线性结构，如线路包含的站点、返回查询的线路结果等等。树形结构没有显式进行使用，但是 Qt 的图形对象都设置了 parent，整个图形界面实际上在逻辑上反映为一颗很大的对象树，MainWindow 是根节点。最后，地铁网络图的表示显然使用了图形结构，Dijkstra、SPFA 等算法也是基于图形结构实现的。

物理结构方面，也是顺序存储、链式存储、散列存储、索引存储均有涉及。前面两项分别对应 QVector 和 QList 的使用，较为普遍。对于需要经常随机访问而很少插入、删除的数据列表而言，使用顺序存储结构；对于需要经常插入。删除的数据而言，使用链式存储结构。由于站点、线路的名称与其本身一一对应，故在地铁系统类中，使用名称到内容的 Hash 表来进行存储。这里用到了散列存储方法，使用了 Qt 中的 QMap 结构。关于线路中包含的所有站点，使用了索引存储结构进行存储。由于地铁系统类中已经存储了站点信息，故线路中不再重复存储，仅记录了其在地铁系统类的 Hash 表中的相应索引，减少了重复存储，提高了效率。

类定义如下：

站点类，记录一个站点的基本信息，如名称、经纬度等等

```
struct Edge{
    QSet<QString> line_list;
    int dist = 0;
};

class Station
{
protected:
    QString name;
    double longi, lati;          //站点经纬度
    QPointF coord;               //站点在图上的坐标位置
    QMap<QString,Edge> edges;    //边
    QSet<QString> lines;         //所属的线路

public:
    Station();

    void latiLongi2coord();
    State addEdge(const Station&,const QString);

    QString getBelongLinesText();

    friend int getDistance(const Station&,const Station&);

    friend class Line;
    friend class SubwaySystem;
    friend class MainWindow;
};
```

线路类，记录一条线路的基本信息，如名称、颜色等等

```
class Line
{
protected:
    QString name;
    QColor color;
    QList<int> total_stations;                  //总站数
    QList<QString> start_stas,end_stas;         //起点站和终点站
    QList<QList<QString>> sta_list;             //站点集合,此处设置是
为了避免一条线有多个分支的情况

public:
    Line(){};

    friend class SubwaySystem;
    friend class MainWindow;
    friend class SubwayControlWindow;
};
```

地铁系统类，综合上面两个类，表示一个完整的地铁系统。查询线路等算法全部在这个类中完成

```
class SubwaySystem
{
protected:
    QMap<QString,Station> stations;     //所有站点
    QMap<QString,Line> lines;           //所有线路

public:
    SubwaySystem();

    State readSubwayFile(QString);
    void statisticEdges();

    State addStation(QString,double longi,double lati,QSet<QString
>);
    State addLine(QString,QColor);
    State addEdge(QString,QString,QString);

    QList<QString> shortTimePath(const QString,const QString);
    QList<QString> lessTransPath(const QString,const QString);

    QList<QString> getSameLineABPath(const QString&,const QString&
)const;
```

```
    friend class MainWindow;
    friend class SubwayControlWindow;
};
```

算法核心类就是上面三个类。另外，我还定义了一些 ui 界面类，便于图形界面绘制。

MyQGraphicsView 类是地铁线路图的绘制窗口的类。这个类的功能比较简单，可以支持绘图窗口的缩放。定义如下：

```
class MyQGraphicsView : public QGraphicsView
{
    Q_OBJECT
public:
    explicit MyQGraphicsView(QWidget *parent = nullptr);

    void zoomIn();
    void zoomOut();
    void resetZoom();
    void refresh();

private:
    void zoom(double);
};
```

SubwayControlWindow 类的作用是显示添加线路/站点/连接的窗口，相应的也有一些方法。定义如下：

```
class SubwayControlWindow : public QWidget
{
    Q_OBJECT

public:
    explicit SubwayControlWindow(int,SubwaySystem*);
    ~SubwayControlWindow();

    void initTab1();
    void initTab2();
    void initTab3();
    void submitTab1();
    void submitTab2();
    void submitTab3();

    void errorNotice(QString);
```

```
    void rightNotice(QString);

private:
    Ui::SubwayControlWindow *ui;

    QString name;
    QColor color;
    SubwaySystem*subsys;
    double longi,lati;

#if USE_CHECKBOX_LIST
    bool* choose_line_list;
    QStringList line_list;
#endif

    void getColor();

signals:
    void done();
};
```

最后，主窗口类的定义如下：

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    void initComboBox();
    void drawSubwaySystem();
    void drawTransLine(const QList<QString>&);

    void queryLine();       //查询线路

    void addStation();
    void addLine();
    void addEdge();

private:
    Ui::MainWindow *ui;
    SubwaySystem subsys;
```

```
    QGraphicsScene *scene;    //绘图场景

    void drawStation(const Station&,const QColor);
    void drawEdge(const Station&,const Station&,const QColor);
};
```

2.5 开发平台

计算机信息：

计算机型号 ： 联想小新 pro13

计算机内存 ： 16.0 GB

处理器 ： AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10GHz

操作系统 ： Windows 10 家庭中文版

开发平台：

编程语言 ： C++（C++11 标准以上）

开发环境 ： Qt Creator 4.15.0，Based on Qt 5.15.2

编译器 ： MinGW 64bit

运行环境：

可以将代码在上述 Qt Creator 集成环境中打开并运行，注意 Qt 版本需要在 5.12 以上。

也可以打开文件夹 5_2_subway_exe，直接运行其中的 5_2_subway.exe，该文件包使用 windeployqt 进行了封装，可以直接在普通电脑上运行。

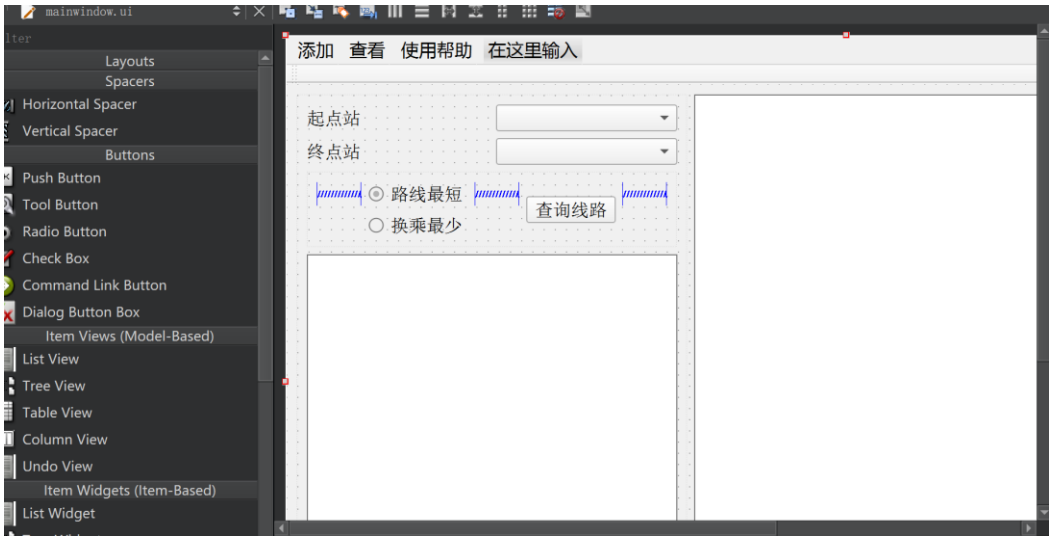
2.6 系统的运行结果分析说明

2.6.1 调试及开发过程

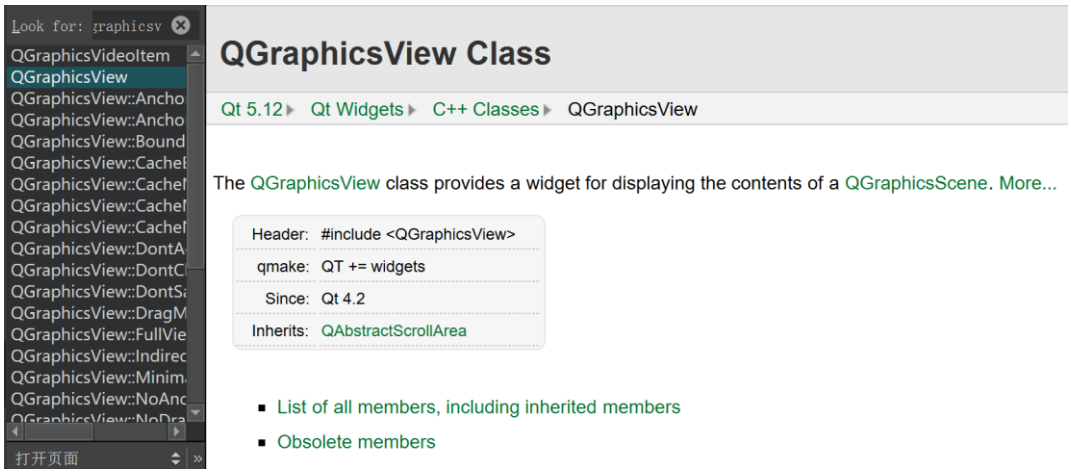
本题相较于上一题，是一个更大的、更综合的过程，调试起来一定也比上一题更难。本题的调试主要通过 qDebug 调试、Qt 自带的 debug 方法调试、以及结果显示到图形界面进行调试三种方法。

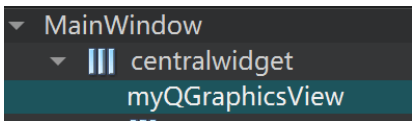
由于本题的核心算法和图形界面完全分开，所以我先编写好这些核心算法的代码，再通过控制台进行算法调试。在算法测试完毕后，再将其综合到图形界面。这样一来，图形界面的各类错误更加容易定位，在调试图形界面的过程中不需要再调试算法部分。

开发过程与上一题类似。由于有了上一题的开发经验，故本题先设计好核心算法，实现好后端的核心代码。然后，再通过代码+ui 拖拽结合的方式设计前端图形界面。如下图：



同时，本题中由于地铁线路显示窗口的图形绘制更为复杂，故我没有继续使用 QPainter 进行图形绘制，而是使用了 QGraphicsView 结合 QGraphicsScene 进行绘图。



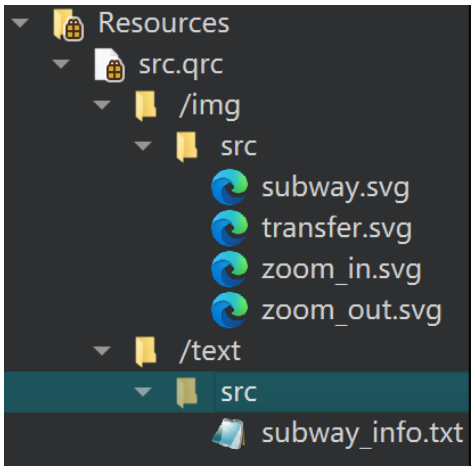



```
private:
    Ui::MainWindow *ui;
    SubwaySystem subsys;
    QGraphicsScene *scene;    //绘图场景
```

如上图，我自己设计了一个可以放大/缩小的 MyQGraphicsView 类，并将 ui 中的 QGraphicsView 提升为我自己设计的类，然后再使用 Scene 进行布局，最终将结果绘制在 MyQGraphicsView 控件上。这种绘图方法比 QPainter 更加灵活，尤其适合用于元素丰富的绘图需求。

另外，本题中使用了一些图像资源和初始化文件资源。Qt 有自己的资源管理方式，使

用.qrc 结合前缀等等进行文件管理，如下图，我创建了一个 Resources 项，并且在其中添加了所需的文件资源信息，同时结合 Qt 自带的 QIcon 以及 QFile 等图形处理/文件处理方式进行使用。




```
QFile file(filepath);
file.open(QIODevice::ReadOnly);
if(!file.isOpen()){
    //报错
    qDebug()<<filepath<<"打开失败";
    return State::ERROR;
}
this->lines.clear();
```

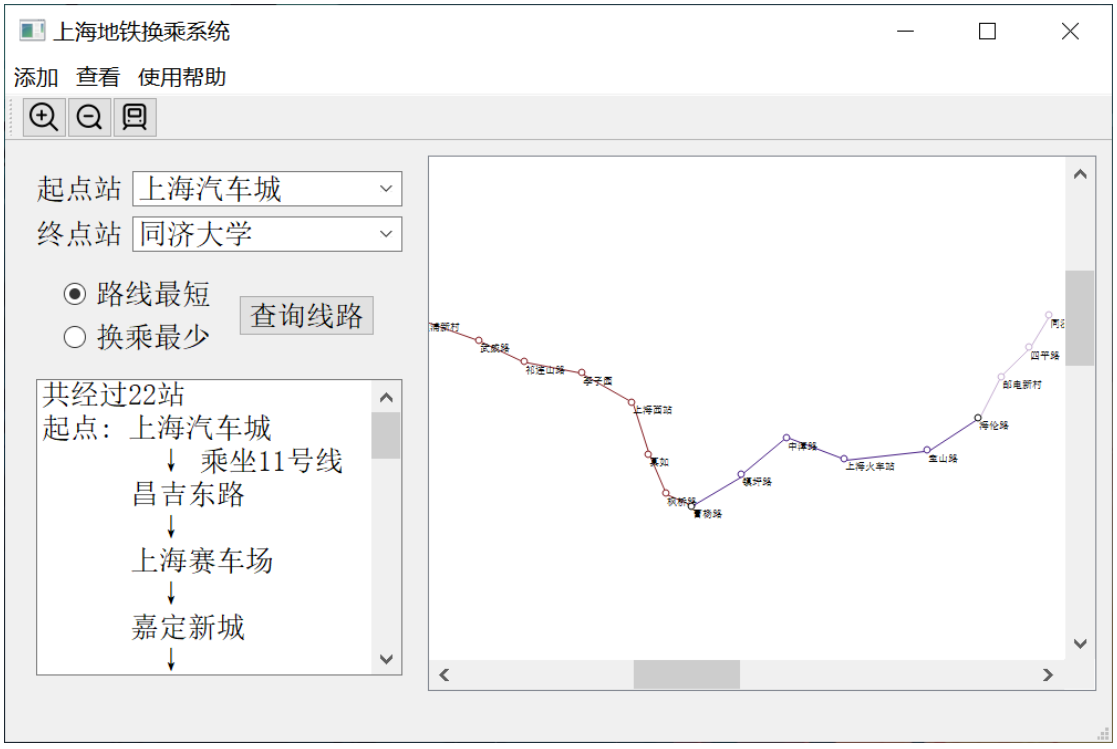
2.6.2 程序正确性展示

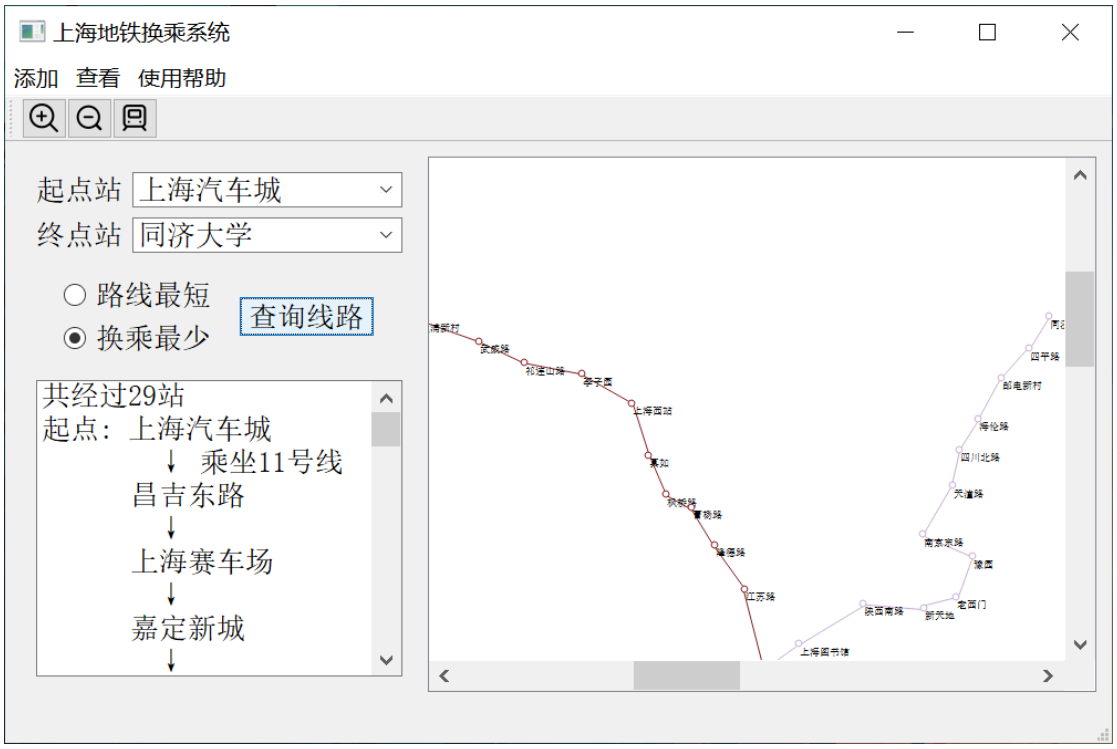
经过多次查询线路、添加站点等操作的检测后，程序表现全部正确。其正确运行界面如下：

程序成功运行后，会自动读取 subway_info.txt 进行初始化，然后在窗口中进行显示。由于初始化的文件读取已经在代码中写好，故这一过程自动完成。如下：

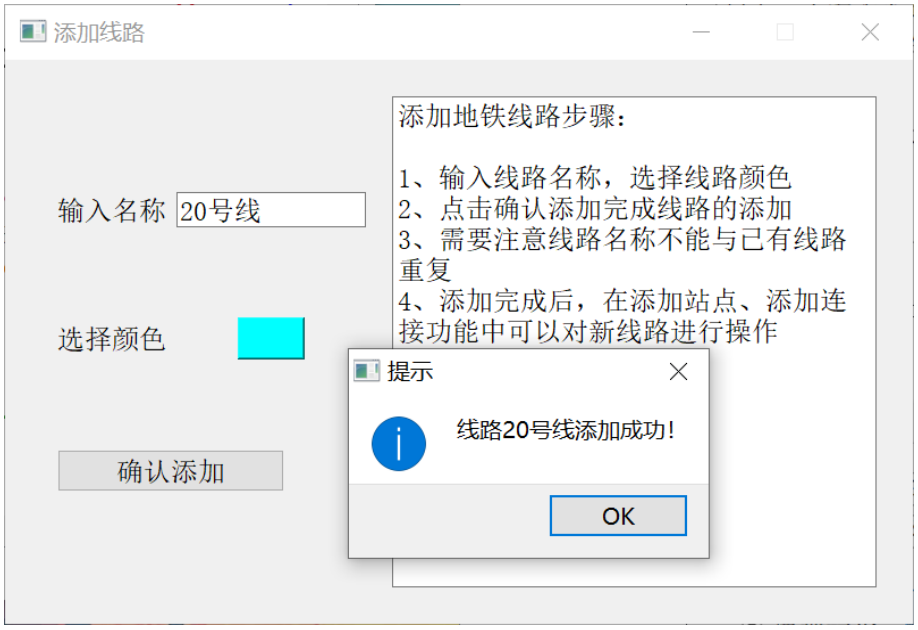
输入起始站和终点站进行查询，如查询上海汽车车到同济大学的最短路线，查询结果如

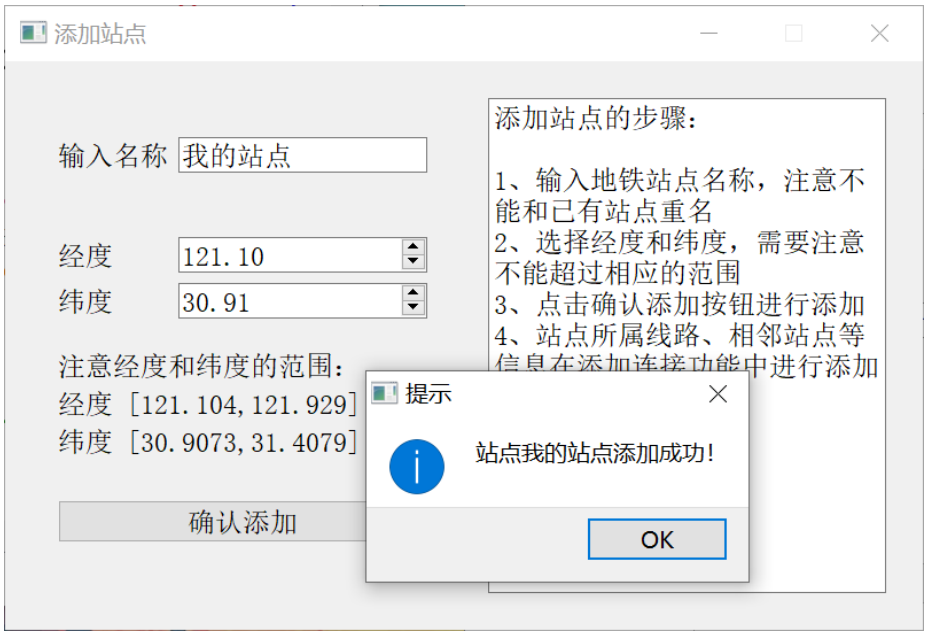
下，左侧显示文字路线，右侧显示图像路线。同样的，可以查询换乘最少线路。可以看到两条路线都是正确的。

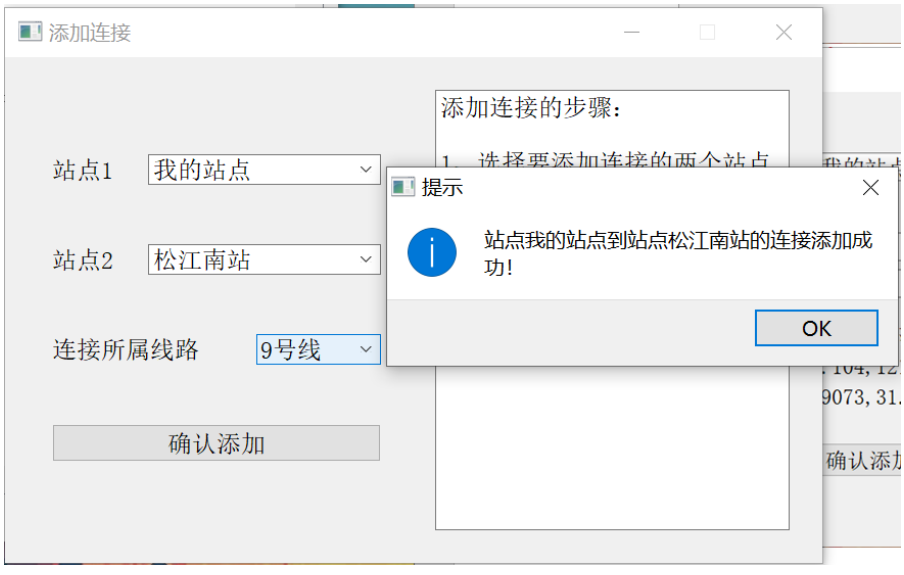


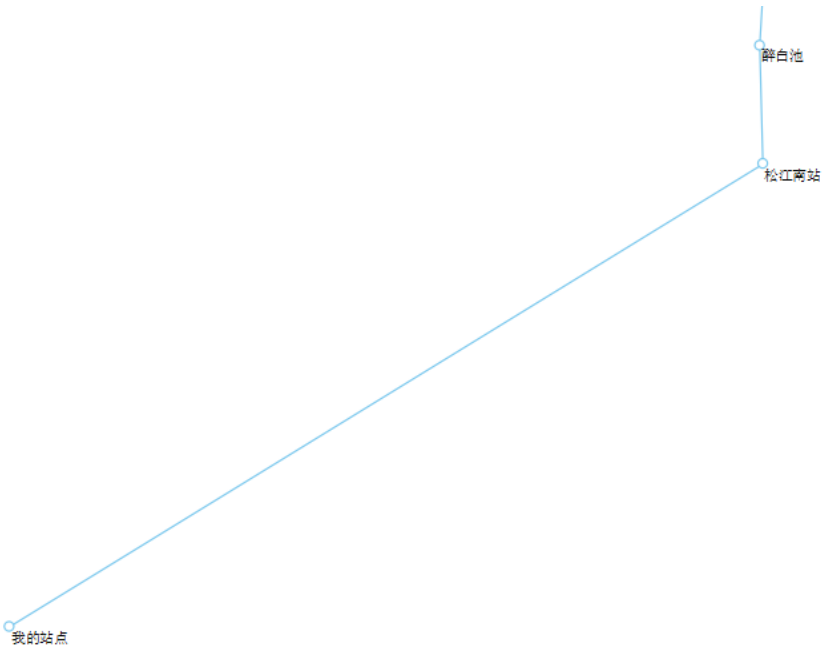


添加线路、站点、连接如下。可以看到添加完站点和连线后，显示窗口会同步进行更新。


添加线路
输入名称
20号线
选择颜色
确认添加
添加地铁线路步骤：
1、输入线路名称，选择线路颜色
2、点击确认添加完成线路的添加
3、需要注意线路名称不能与已有线路重复
4、添加完成后，在添加站点、添加连接功能中可以对新线路进行操作
提示
线路20号线添加成功!
OK


添加站点
输入名称
我的站点
经度
121.10
纬度
30.91
注意经度和纬度的范围：
经度 [121.104, 121.929]
纬度 [30.9073, 31.4079]
确认添加
添加站点的步骤：
1、输入地铁站点名称，注意不能和已有站点重名
2、选择经度和纬度，需要注意不能超过相应的范围
3、点击确认添加按钮进行添加
4、站点所属线路、相邻站点等
提示
站点我的站点添加成功!
OK


添加连接
站点1
我的站点
站点2
松江南站
连接所属线路
9号线
确认添加
添加连接的步骤：
提示
站点我的站点到站点松江南站的连接添加成功!
OK



2.6.3 程序稳定性展示

程序的稳定性表现优秀。所有的类中使用 new 运算符动态申请的内存，全部在析构函数中进行了 delete 操作。图形界面中动态申请的 Qt 类对象，全部设置了 parent。在 parent 关闭时，这些 new 运算符申请的 Qt 类对象会自动进行析构。

各类消息对话框全部使用 Qt 自带的 QMessageBox 进行，避免在主窗口关闭前反复触发消息对话框可能导致的内存耗尽情况。

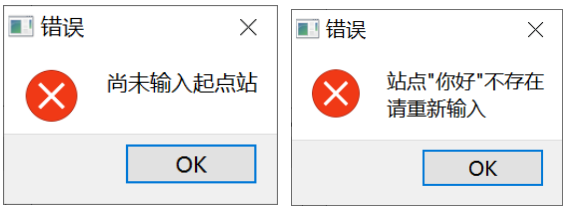
另外，每次进行添加操作后，如果图形界面有变化，都会进行 update 操作，进行同步更新。

```
void MyQGraphicsView::refresh()
{
    this->viewport()->update();
}
```

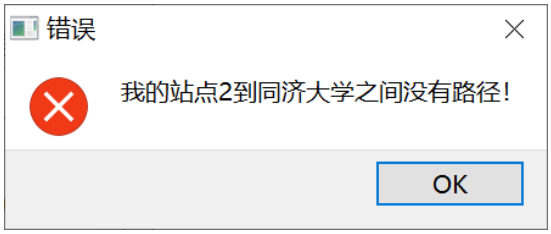
2.6.4 程序容错能力展示

程序对各种错误都进行了详细的、周全的处理。如下：

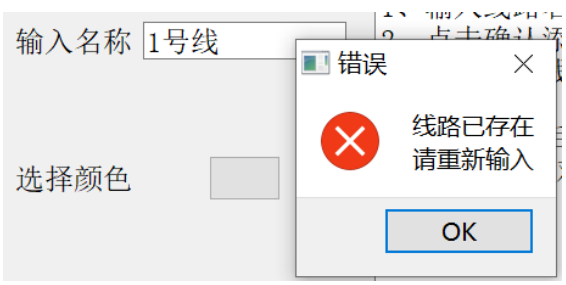
查询的起点/终点不存在，会弹出对话框进行提示。

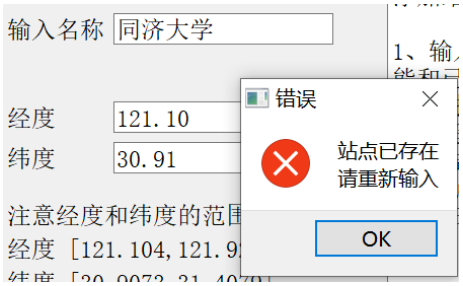


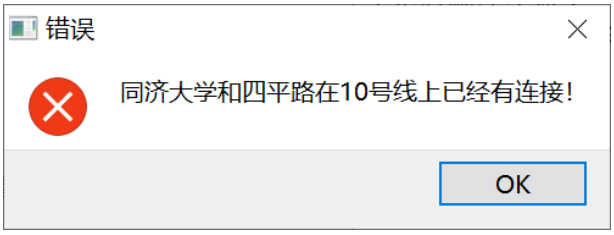
对于新添加的站点，可能出现起点站和终点站之间没有线路的情况。这种情况也有提示。



添加线路、站点等重复，会有提示。



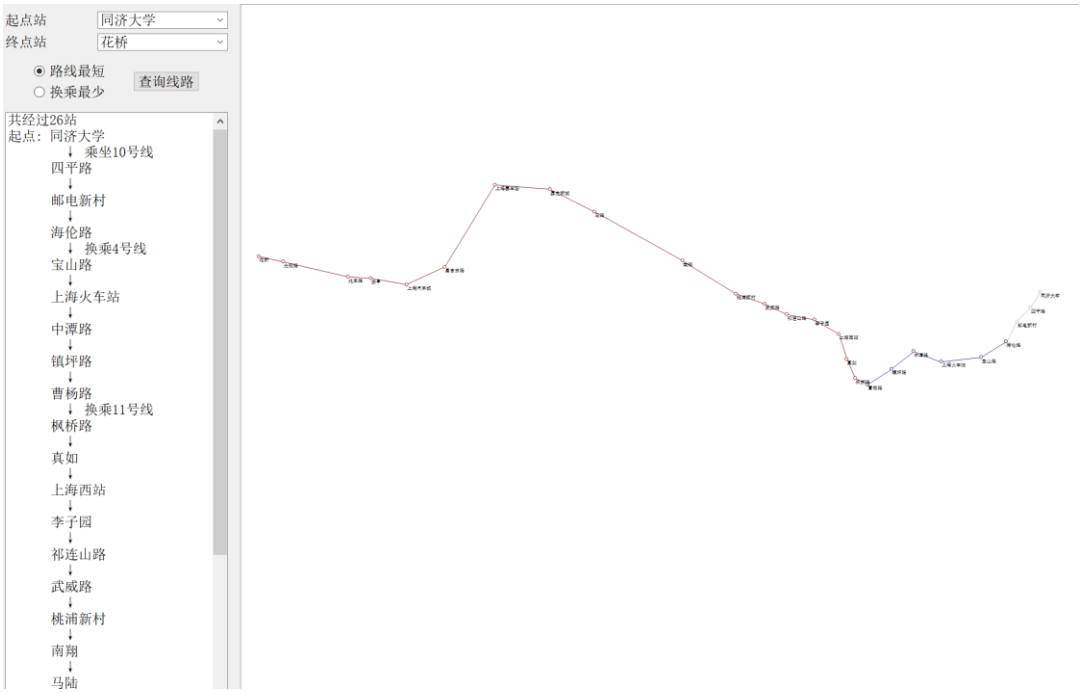




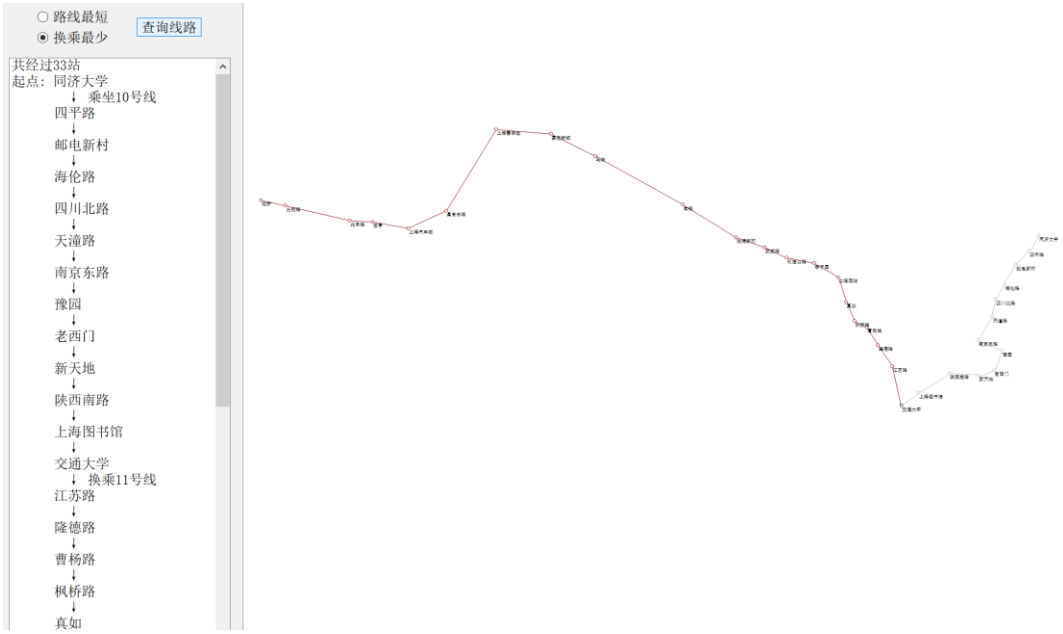
还有一些别的错误处理，此处不再列出。

2.6.5 运行案例说明

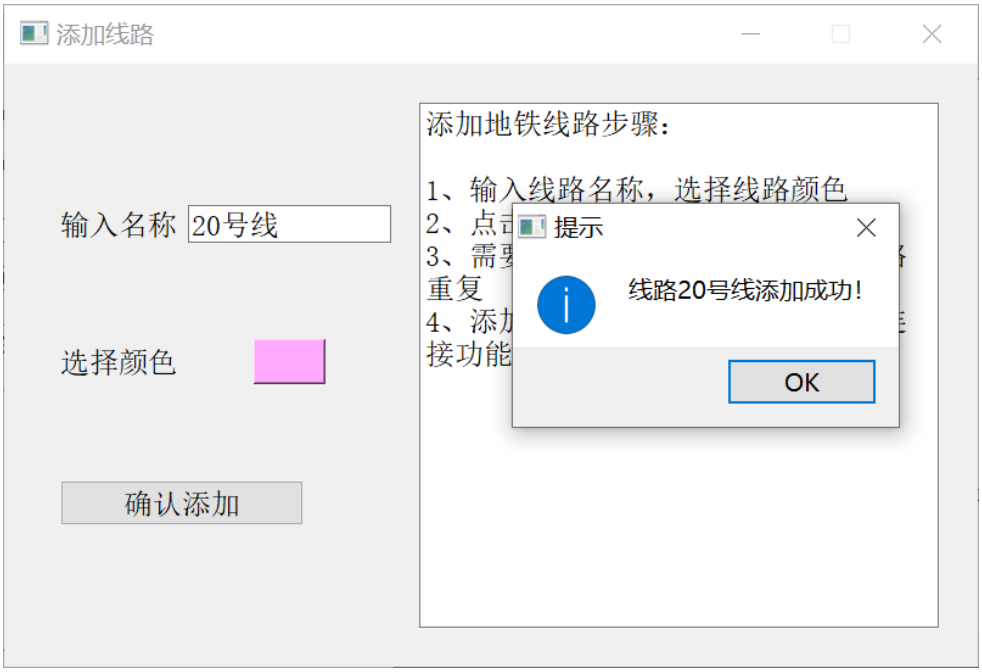
起始站点输入同济大学，终点站输入花桥，查询最短路线：



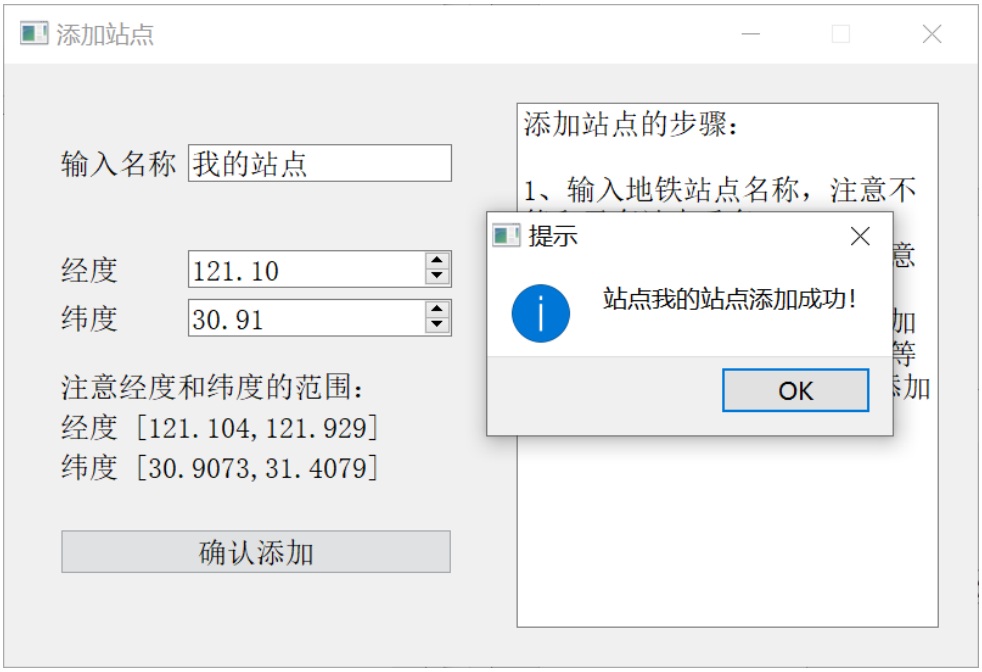
切换到最少换乘，查询路线：



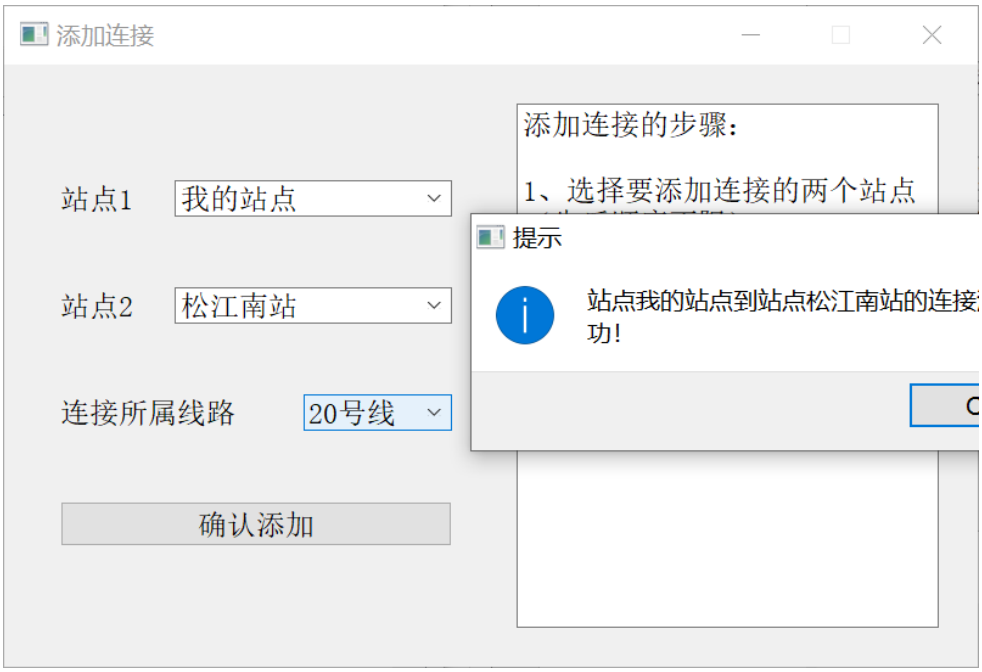
添加“20 号线”，并选择颜色为粉色：



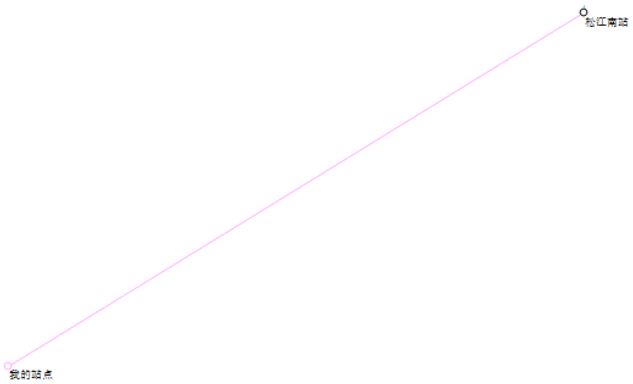
添加“我的站点”：



在“我的站点”和“松江南站”之间添加 20 号线的连线，成功

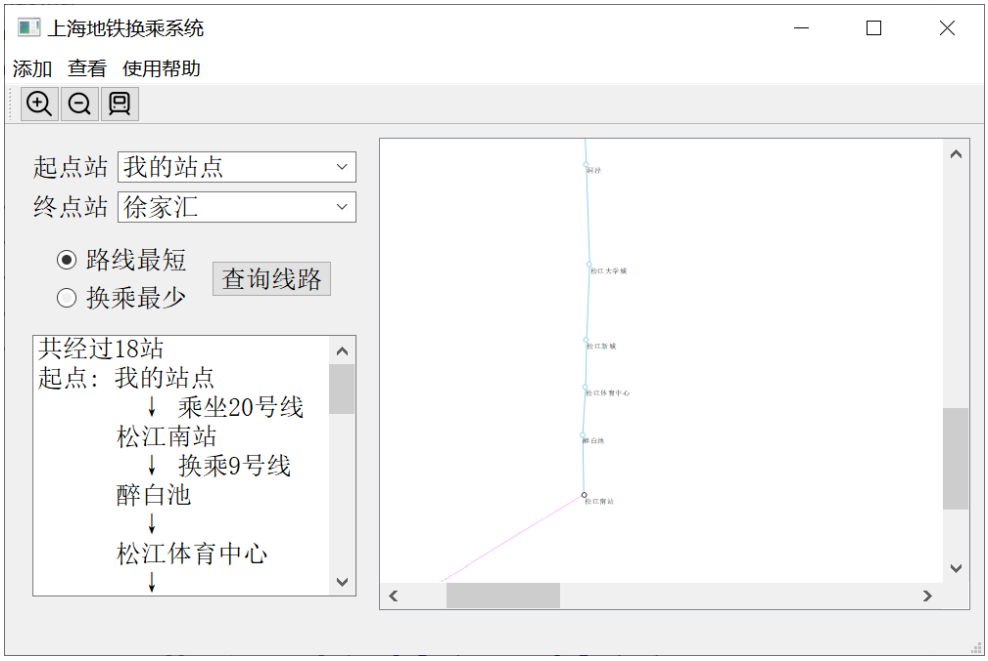


可以看到图上也出现了一条连线。



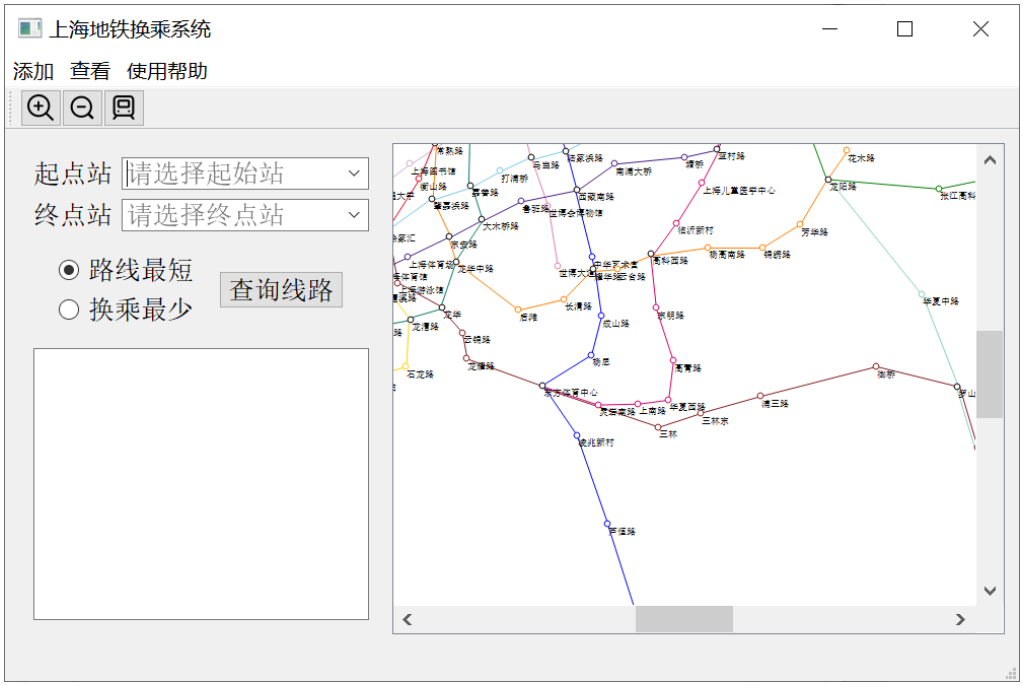
案例中有一处需要注意。添加线路完成后，这条线路上不含任何的站点。添加站点完成后，这个站点也不属于任何线路，同时也不和任何其他站点相连。只有在添加连接后，被添加的站点自动加入到线路上。

查询我的站点到徐家汇之间的路线，可以发现新添加的“我的站点”和其他站点具有相同的作用：

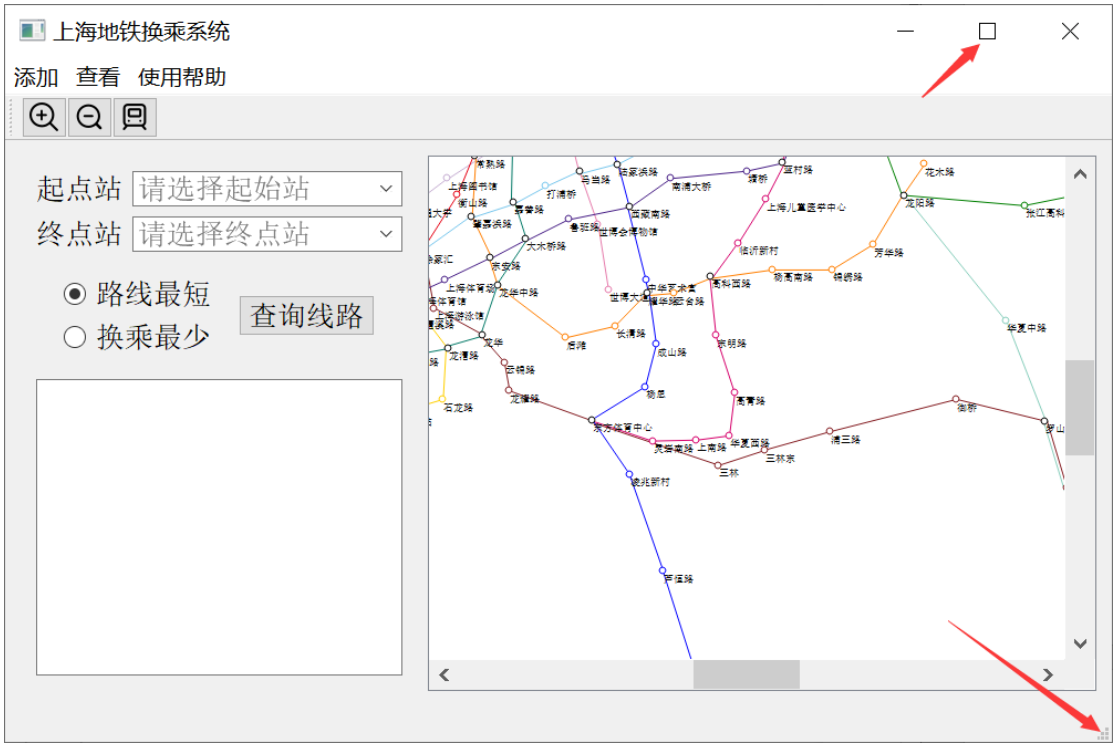


2.7 操作说明

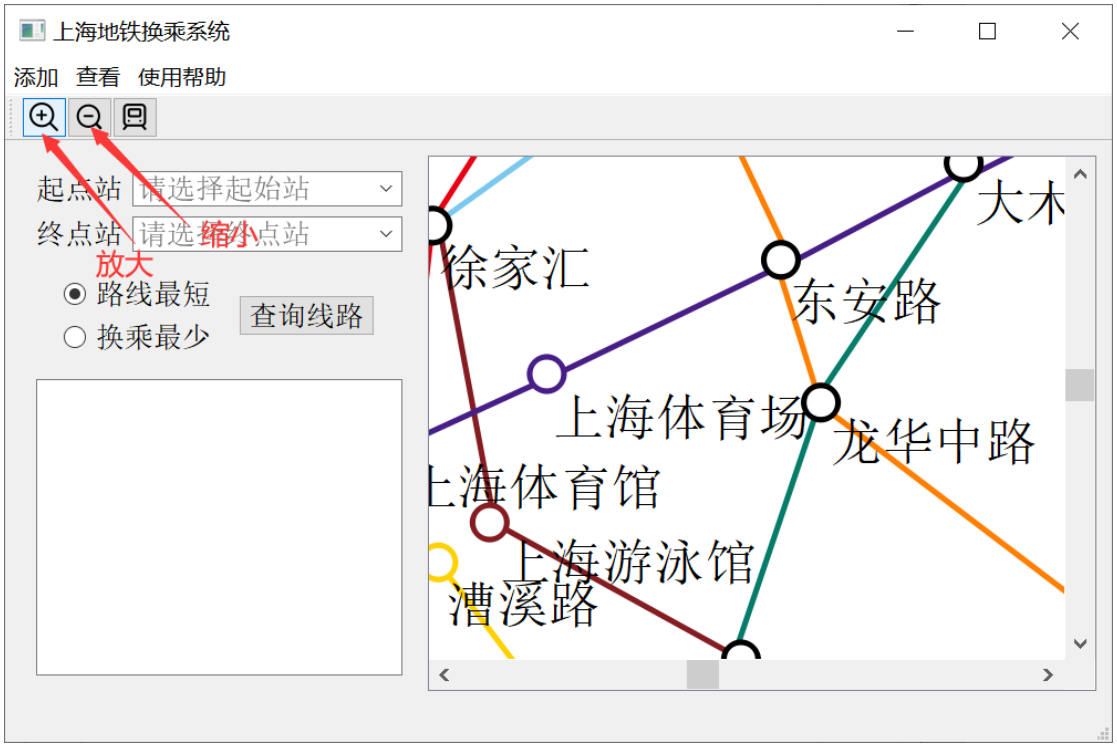
程序运行后，展示如下界面：



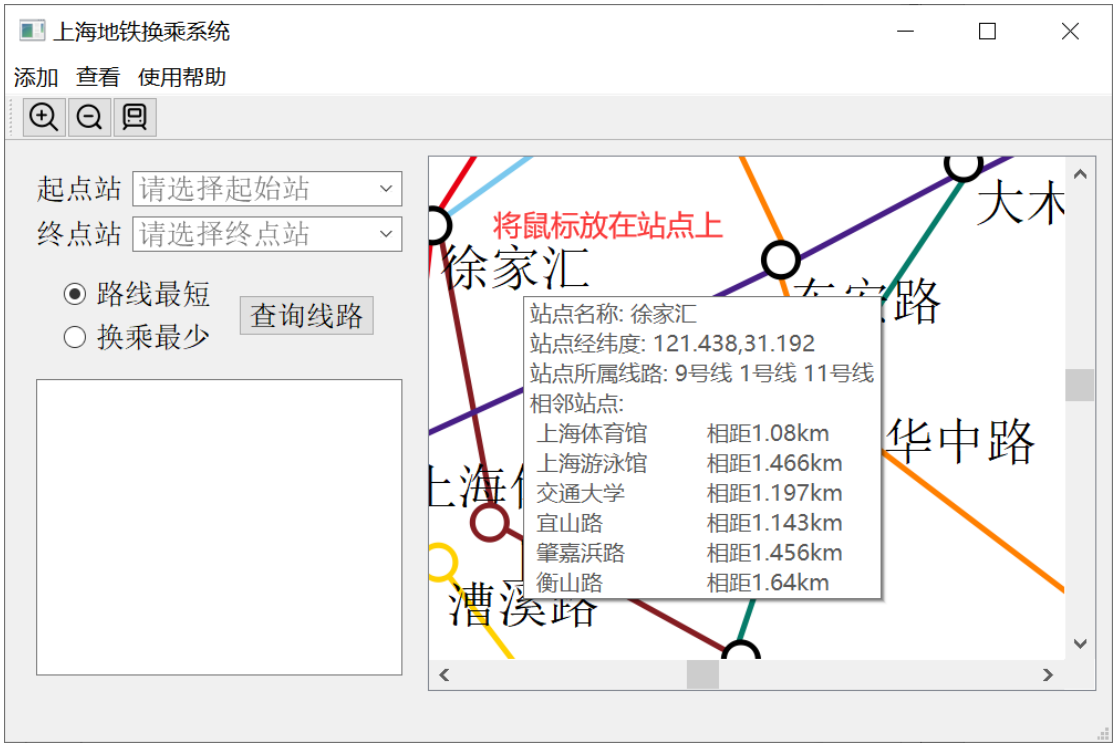
在右侧可以看到上海地铁线路图。其文件信息存储于 src 目录下的 subway_info.txt 中。可以拖拽右下角或者点击最大化按钮来获得更大视图。



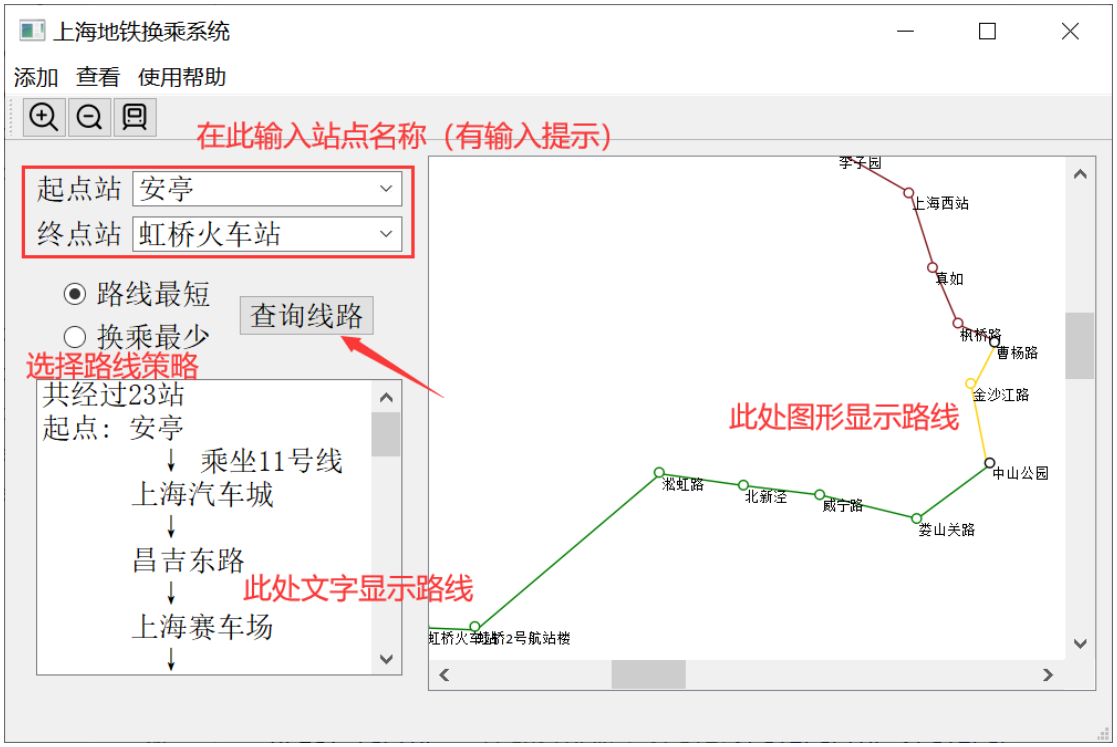
可以通过左上角的放大镜图标来进行地图放大和缩小。放大和缩小以当前窗口的中心为中心进行。也可以通过“查看”栏中的放大、缩小来进行同样的操作。



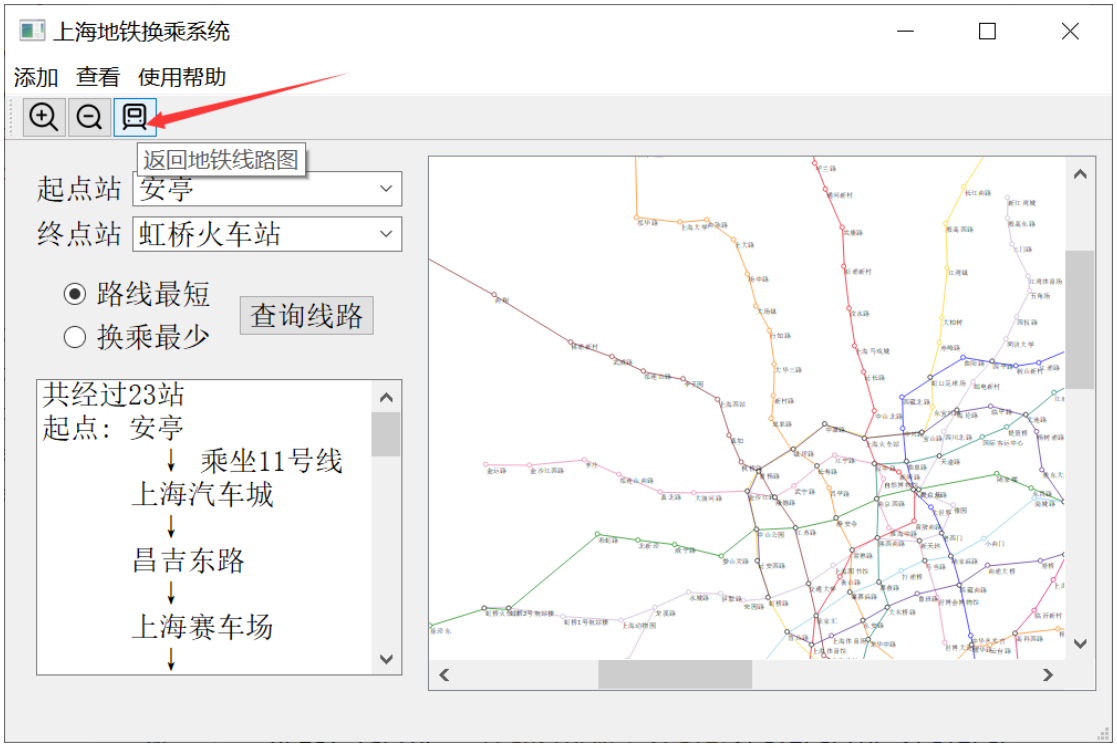
将鼠标放在站点上，可以显示出有关信息。



在左边的输入框中输入起点站和终点站，选择查询策略，可以查询线路。



此时，点击左上角的地铁图标，或者点击“查看”中的“返回地铁线路图”可以返回整个地铁网的图形界面。



点击“使用帮助”中的帮助，可以弹出一个简易的使用提示