华中科技大学网络空间安全学院

程序设计综合课程设计报告

题目：武汉地铁乘车路线推荐系统

班　　级：网安1902班

学　　号：U201911808

姓　　名：袁　也

成　　绩：

指导教师：肖　凌

完成日期：2021年3月20日

目 录

一、系统需求分析

1.1 背景介绍

随着城市的发展，越来越多的城市都修建了自己的复杂的地铁网络。然而，由于计价、路线规划等需求的产生，需要通过计算机程序来实现一些地铁线路图上的算法和功能，来方便人们出行。本次程序设计基于这样的背景，希望能够实现一个比较完善的系统，通过数据模拟的方式来验证系统的正确性、可行性和鲁棒性。此外，还在程序中预留了进一步升级优化的接口，可以在今后接入更加全面的真实地铁数据，从而运用到生产实践之中。

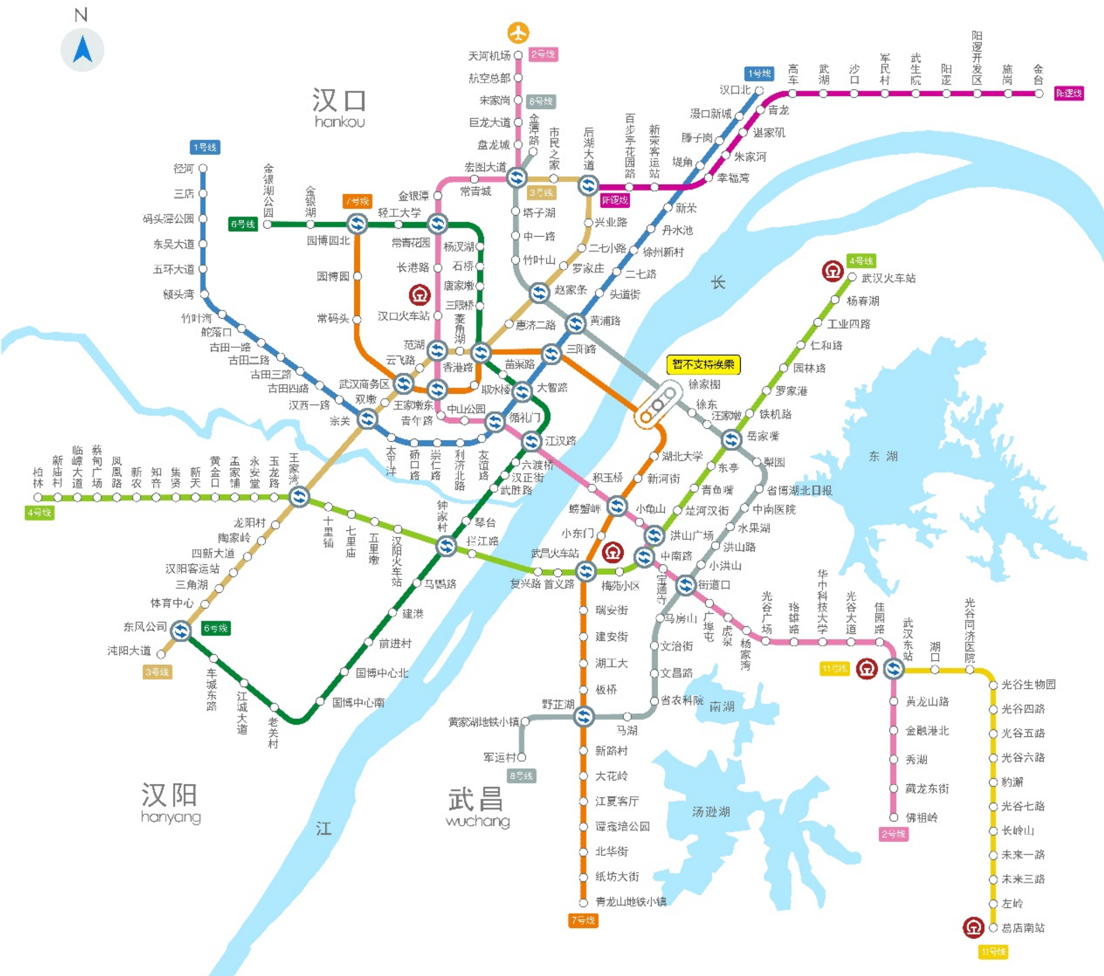

图 1-1 武汉地铁线路图

1.2 研究现状

当前，市场上已经有成熟的地铁线路导览系统，其代表有百度地图、高德地图等，他们基于网页实现，如图 1-2 所示为百度实现的武汉地铁线路导览图。其主要有线路图展示、时刻表查询、路线规划、首末站查询等功能，还有漂亮的图形化界面。这些功能大多都有较大的难度，因此百度地图的武汉地铁查询系统仅作为功能上的参考，在我的课程设计程序内，将会对这些功能做出一定的简化再编写程序。

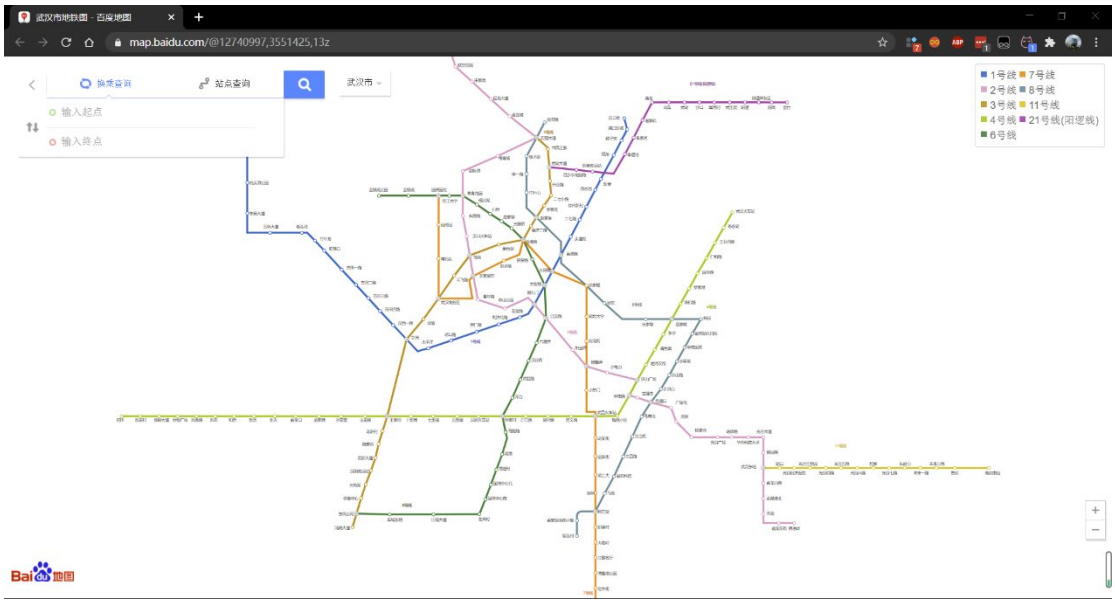

图 1-2 百度地图武汉地铁官网导览图

1.3 要解决的问题

首先，根据题目要求，我们对问题进行分析及拆解划分：

1）构建模拟的武汉地铁主要 7 条已开通线路和站点；

解析：首先，将武汉地铁主要 7 条已开通的线路全部信息整理成一份具有标准格式的.json 文件，其中具有线路名称、线路颜色值、线路长度、线路最大载客量以及线路车站数，紧跟着车站列表，每一座车站包含车站经纬度坐标、车站名称、车站是否为换乘站、车站可换乘线路（如果有）、车站距离下一站的距离、车站到下一站的拥挤度等，此外，还有票价计费规则、运营时刻表等其他地铁系统信息。由于.json 文件的易扩展

性，我们很容易在有新线路建设完成后加入新的线路信息，以及随时扩展各种各样的数据域。

然后，我们需要设计一套文件读取的函数，并设计相应的数据存储结构体，从而将外存中的配置文件按照特定的结构读入到内存当中暂存。

最后，根据之前内存中暂存的线路数据，构建邻接表，用来表示地铁线路图的信息。

2）构建模拟的地铁线路运行、乘车、换线等基本功能；

解析：运用存入内存的静态信息（类似数组或链表等结构存储），设计一套函数来手动遍历地铁网络。同时，能够识别出几种不同的车站进行不同的处理：中间普通车站、中间换乘车站（可能含有多条换乘线路）、端点普通车站、端点换乘车站等，换乘车站应该能够运用某种方法将搜索指针转移到另一条线路的数组或链表之上。

3）采用对无向有权边的图顶点间最短距离的计算，提供地铁换线转乘的路线推荐建议，包含 1-3 条可行的线路及换乘、路途时间、票价金额等信息；

解析：基于在步骤（1）构建的邻接表，设计图搜索函数（例如 DFS、Dijskra、A*算法），搜索出多条路线暂存在内存当中，然后通过函数分析每一条路线的不同属性和特征，最后利用一定的排序算法将暂存的全部路线按照设定规则排序后输出，可能还会加入一些筛选条件。

4）扩展实现：引入本城上下班、购物、娱乐以及跨城市交通等线路类型的流量模拟，形成不同时段的线路乘车人流量拥挤程度状态；在此基础上，分析地铁人流量的拥挤程度对于转乘建议的影响。

解析：通过一个全局标志变量来判断当前是否为手动改变的人流量拥挤度，假如判断到当前为手动改变人流量拥挤度，则通过一个函数对内存之中存储的信息进行暂时的修改，此时需要注意不能修改配置文件 config。然后在此基础上，进行多次模拟实验，观察寻路算法是否能够真正识别到线路拥挤度的变化，从而设法绕过过度拥挤的区间。

1.4 输入数据描述

输入数据应当分为三部分：服务端通过互联网传来的后端配置文件数据、客

户端可交互界面输入的始末车站等用户需求以及通过配置文件形式存放在本地文件夹的用户设置配置文件。

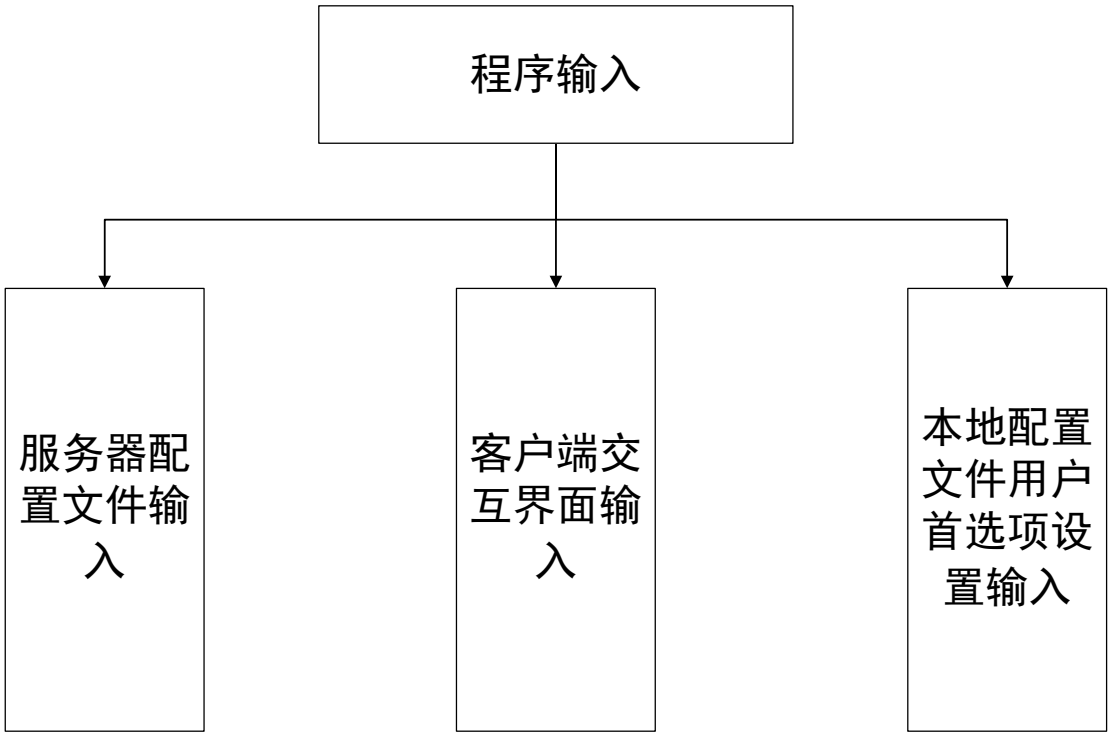

图 1-3 程序输入结构图

关于地铁线路网信息的后端配置文件数据输入已在前面一节做了详细阐述，在此不再赘述。

用户通过客户端的交互界面选择起点站和终点站，通过内置函数在数据结构中寻找到对应车站的结点；然后，用户会输入路线规划的模式（综合排序模式、时间最短模式、里程最短模式、票价最低模式、换乘次数最少模式以及舒适度优先模式等），由于本程序需要模拟各个时间点的路线情况，因此还需要用户通过可视化交互界面输入设定的时间等信息。此外，由于程序需要做成可视化交互界面的形式，因此用户也会通过触发动作的方式向程序输入不同种类的信号，依然需要通过代码进行解释和逻辑判断。

通过配置文件的用户首选项设置输入，程序将为用户做出个性化的改变。此外，配置文件还可以暂存上一次用户关闭程序之前的部分数据，从而帮助用户在上次打开程序规划路线的基础上继续规划路线。

1.5 程序输出描述

程序需要将时刻表、规划的路线等信息以可视化交互界面的形式组织在软件的界面中。

关于时刻表。程序需要显示当前设定的时间，并在图形化界面上展示下一班车到站的时间。

关于规划路线信息。首先程序需要输出起点站和终点站信息，然后输出各条路线。每条路线首先需要输出路线的参数，比如路线长度、整体拥挤度、换乘次数、用时、票价等信息。然后需要以遍历车站的形式将这条路线上经过的每个车站依次输出。路线输出时应当按照用户的首选设置进行筛选及排序等操作，此外，路线输出数量应当不少于三条。

此外，交互式界面还要求及时输出错误信息及各组件的实时状态。这些将基于选定的图形化界面设计框架进行具体实现。

1.6 预期设计目标

用户从互联网上得到本程序的安装包后，可以按照安装程序的提示一步一步完成安装和配置。运行程序后，程序有指引性地引导用户输入所需数据，并选择需要的操作，然后能够正确打印用户所需的路线信息。此外，程序还应提供手动遍历地铁线路图的功能，使用户能够通过上一站、下一站、换乘线路这三个指令来完成遍历地铁路线操作。

程序还需要预留今后进行优化提升的空间。比如，预留配置文件通过网络进行输入的输入接口，真实地铁线路图输入输出的功能接口以及绘图等操作实现的必要准备。

二、总体设计

地铁路线导览系统框架如图 2-1 所示。

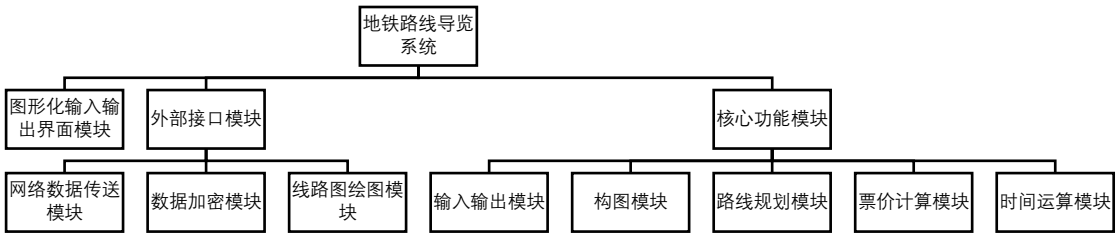

图 2-1 地铁路线导览系统框架图

注：上图所含模块为系统设计的全部模块，根据本次课设的要求，代码仅实现了部分模块和功能，但足以满足课程设计的要求。

2.1 图形化输入输出界面模块

程序采用 Qt 框架实现图形化输入输出界面功能。Qt 是基于 C++的面向对象的框架，使用特殊的代码生成扩展（称为元对象编译器(Meta Object Compiler, moc)）以及一些宏，Qt 很容易扩展，并且允许真正地组件编程。

图形化输入输出界面应当充当好以下几个角色：输入输出过滤器、与用户交互的小助手、引导用户操作。

图形化界面应当从用户角度出发，设计出人性化的界面。

2.2 外部接口模块

2.2.1 网络数据传送模块

这个模块起到端到端的数据传输功能，应当基于当前流行的网络数据传送模块，例如 Socket 协议构建。这个模块还应考虑到数据校验及其他安全性问题。这个模块暂时没被包含在本次课设的程序设计当中。

2.2.2 数据加密模块

数据加密模块将网络传来的数据进行解密，同时将导出的文件等进行加密。

数据加密解密模块采用 OpenSSL 库。

2.2.3 线路图绘图模块

线路图绘图模块根据核心模块输出的车站坐标、路线详细信息等数据，基于某地图 API 进行绘图工作。然而这部分暂时还在调试，因此没有体现在课程设计的程序当中。

2.3 核心功能模块

2.3.1 输入输出模块

输入输出模块作为衔接图形化交互界面与程序算法核心的桥梁，应当具备数据过滤、数据验证、异常抛出、易扩展、强兼容等特点。输入输出模块一端连接图形化界面，主要由 C++进行实现，基于面向对象编程思想。另一端连接由 C 语言实现的基于过程的算法核心，对数据的格式及形式要求严格。因此，设计的输入输出模块应当使数据“宽进严出”。

此外，输入输出模块还承担着将加密解密模块输出的文本文件进行解析和读取，并将其中的数据按照程序定义的结构体，有结构有条理地存入计算机的内存当中，并在内存中划出一块特定的空间用于存储这些原始数据，以便后面的其他功能随时调用。

2.3.2 构图模块

构图模块根据输入输出模块在内存中存储的结构化的全部数据，生成一个邻接表，存储整个地铁网络的全部信息。此外，还将在起始车站和终到车站选定后生成一份仅包含换乘站和始末车站的换乘子网，用以简化计算量，加快运算速度。

2.3.3 路线规划模块

路线规划模块根据换成子网，使用 DFS 算法按照特定的最大搜索深度搜索全部可行路线，并将这些路线全部暂存在内存中。而后根据路线评估函数完善路线的有关属性信息，如整体拥挤程度、路线耗时、换乘次数、里程、票价等信息。

最后需要使用特定的排序项进行排序，最后在输出到输入输出模块进行用户交互展示。

2.3.4 票价计算模块

票价计算模块较为独立，可以单独使用，也可以与其他程序模块耦合使用。票价计算模块根据输入输出模块从配置文件读入的票价计算方法进行计算。计算完的结果返回到输入输出模块向用户展示。

2.3.5 时间运算模块

时间运算模块根据输入输出模块从配置文件读入的拥挤度信息和时间系数等计算路线的实际需要时间，计算结果返回到调用处。

三、数据结构设计

3.1 结构体定义

3.1.1 车站编号

```
1   struct StNum {
2       int line;
3       int station_number;
4   };
```

车站编号结构体定义如上面的代码块所示，其中包括线路信息和车站序号。

3.1.2 车站信息

```
5   struct Station {
6       StNum id;
7       string name;
8       bool transfer;
9       vector<int> TransTo;
10  };
```

车站信息结构体定义如上面的代码块所示，其中包括车站编号结构体、车站名称、是否为换乘车站以及可以换成的线路编号（如果有的话）。

3.1.3 全图邻接表边结点

```
11  struct StNode {
12      Station station;
13      StNode* next;
14  };
```

全图邻接表边结点结构体定义如上面的代码块所示，其中包括车站信息结构体和下一站指针。

3.1.4 全图邻接表车站静态数组

```
15  struct StArray {
16      Station station;
17      int numOfAdjacentNodes;//邻接节点数量
18      StNode* next;
19  };
```

全图邻接表车站静态数组结构体定义如上面的代码块所示，其中包括车站信息结构体、邻接结点数量和下一站指针。

3.1.5 线路信息

```
20  struct Line {
21      int id;
22      int fullNum;
23      double length;
24      int St_Num;
25      int cong_flag;//手动拥挤度设置模式：0~预设拥挤度，1~手动宽松，2~手动一
    般，3~手动拥挤
26      vector<Station> st_list;
27  };
```

线路信息结构体定义如上面的代码块所示，其中包括线路编号、线路最大承载人数、线路全长、车站数量、拥挤度标志和车站列表等信息。其中拥挤度标志提供手动拥挤度设置模式：0~预设拥挤度，1~手动宽松, 2~手动一般, 3~手动拥挤。

3.1.6 HH:MM:SS 时间

```
28  struct TimeHMS {
29      int h;
30      int m;
31      int s;
32      TimeHMS(){}
33      TimeHMS(int _h, int _m, int _s) :h(_h), m(_m), s(_s) {}
34  };
```

HH:MM:SS 时间结构体定义如上面的代码块所示，其中包括时、分、秒信息，以及两个构造函数 TimeHMS(){}和 TimeHMS(int _h, int _m, int _s) :h(_h), m(_m), s(_s) {}，用于更加方便地初始化结构体变量。

3.1.7 路线信息

```
35  struct Route {
36      int price;
37      double dist;
38      int time_sec;//考虑拥挤后的时间
39      int transfer_times;//换乘次数
40      int s2s_traffic[3];//区间拥挤情况
41      double overall_traffic;//路线综合拥挤程度
42      double weightedScore;//加权分数
43      vector<StNum> way;
44  };
```

路线信息结构体定义如上面的代码块所示，其中包括路线票价、路线里程、考虑拥挤度影响后的真实时间、路线换乘次数、区间拥挤情况统计数组、路线综合拥挤程度指数、加权分数以及由车站编号构成的路线结构体数组。一个 Route 结构体能够存储一条路线，在使用时开一个静态数组，即可存储 DFS 生成的每一条路线。

3.1.8 换乘网邻接表车站结点

```
45  struct TransferNode {
46      StNum st;
47      bool isTransferEdge;
48      //isTransferEdge==false
49      int count_station;//两站间站数(包括这两站中的一站)
50      double dist;//两站间距
51      int time_sec;//两站间耗时(原始时间)
52      //isTransferEdge==true
53      int transfer_time_sec;//换乘时间
54      TransferNode* next;
55  };
```

换乘网邻接表车站结点结构体定义如上面的代码块所示，其中包括车站编号、是否为换乘边、两站间车站数量、两站间距、两站间原始耗时（未用拥挤度进行真实时间折合）、换乘时间（如果这条边是换乘边）和下一个换乘节点指针。

换乘网中仅包含地铁网络中全部换乘车站以及用户选定的始末车站，对于每一个换乘车站，系统将每一个站台都当作一个车站结点。例如循礼门站是 1 号线和 2 号线的换乘车站，则有两个节点表示循礼门站。这两个结点在数据域中有所区别，它们的车站编号不同。

为了区分换成站内不同站台之间的边和不同换乘站之间的边，在换乘节点中增设了“isTransferEdge”这个数据域，假如为真，则代表这条边两端的结点为同一座车站的不同线路站台；反之，为假则说明这条边连接的是两座不同的车站。

3.2 全局变量定义

由于 Qt 是面向对象的图形化界面，然而核心确实主要由 C 语言实现的面向过程思想编写的程序，因此需要一些必要的全局变量用于模块之间的数据传输。

3.2.1 线路信息数组

线路信息数组由线路信息结构体构成的静态数组（见 3.1.5），将地铁系统中全部线路的线路信息及车站链表存储在内存当中，其定义如下：

```
56  extern Line line[MAX_LINE_NUM];
```

同时伴随线路信息数组的，还有一个布尔型的开通线路数组，也就是一个哈希表，用以判断一条线路是否开通，其定义如下：

```
57  extern bool openline[MAX_LINE_NUM];
```

3.2.2 全图邻接表

全图邻接表由一个全图邻接表车站静态数组构成（见 3.1.4），每一个数组结构体都向外引一条邻接链表，表示与这个车站相邻接的车站列表，其定义如下：

```
58  extern vector<StArray> AdjLst;
```

静态数组借助 vector 实现，vector 本质上和静态数组类似，但是比静态数组更加节省空间，因此选择使用 vector 进行优化。全图邻接表的可视化结构如图 3-1 所示。

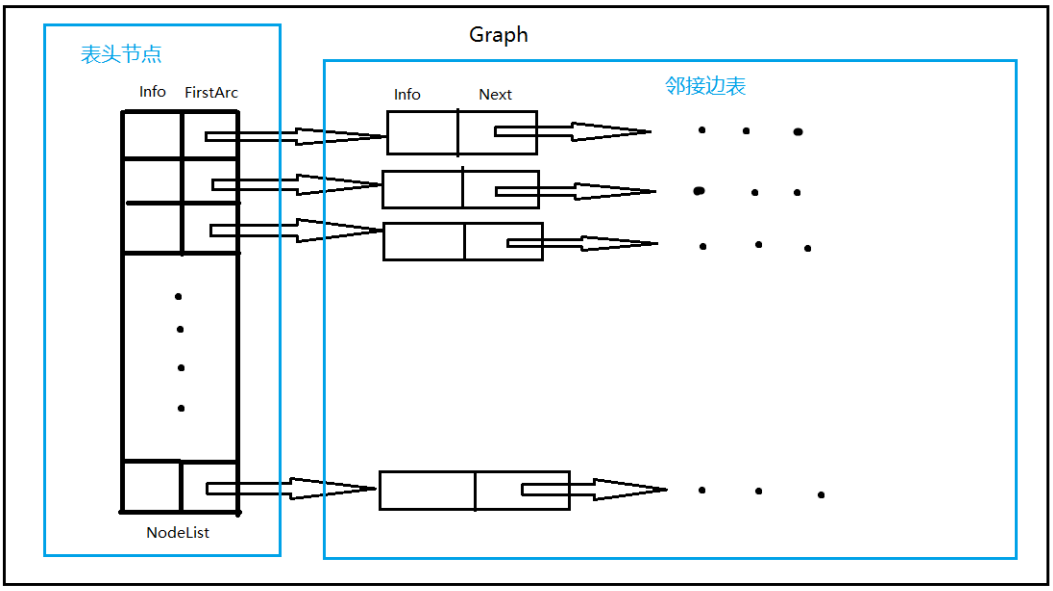

图 3-1 全图邻接表结构

3.2.3 DFS 访问数组

DFS 访问数组是 DFS 算法的配套用品，当 DFS 算法寻路过程中，需要通过车站的编号来检索判断这座车站是否被访问过。假如访问过，则不再访问这座车站，否则将这座车站纳入到下一个递归访问的结点集当中。因此，设计 DFS 访问数组的全局变量定义如下：

```
59  extern bool vis[MAX_LINE_NUM][MAX_STN_NUM];//DFS 访问表
```

不难看出，这个数组有两个维度，第一个维度是车站所在的线路，第二个维度是车站在这条线路上的编号。由于地铁中的任意一个站台和车站编号成一一对应的双射关系，因此可以引入这样一个访问哈希表进行标记。

3.2.4 可行路线数组

由于 DFS 使用递归实现，因此当 DFS 每得到一条可行路线之时，都需要将这条路线存储在一个全局变量数组中。可行路线数组的每个元素都是一个路线结构体（见 3.1.7），其定义如下：

```
60  extern vector<Route> R;
```

3.2.5 换乘网邻接表

考虑到地铁线路网的自身特点，即大部分站点都是非换乘车站，其邻接车站只有一站或两站，而换乘车站数量较少，占比很低，因此整张网络的连通程度较低。如果保持原始网络进行 DFS，可能会造成较大的资源浪费却仍难以提升算法效率。因此，结合地铁线路网的特点，我将地铁整个网络简化为仅含换乘车站和用户选定始末车站的简化换乘站子网，并在这个子网上跑 DFS 算法，可以大大降低运算量。

于是，需要开辟一个全局变量空间，用来存储换乘网邻接表，其定义如下：

```
61  extern vector<TransferNode> TN;
```

这是一个数组，其中每个元素都是一个换乘网邻接表车站结点，具体见 3.1.8，每两个换乘车站之间的全部车站和线路信息全部抽象为每个换乘站结点当中的数据域信息。换乘网邻接表结构如图 3-2 所示。

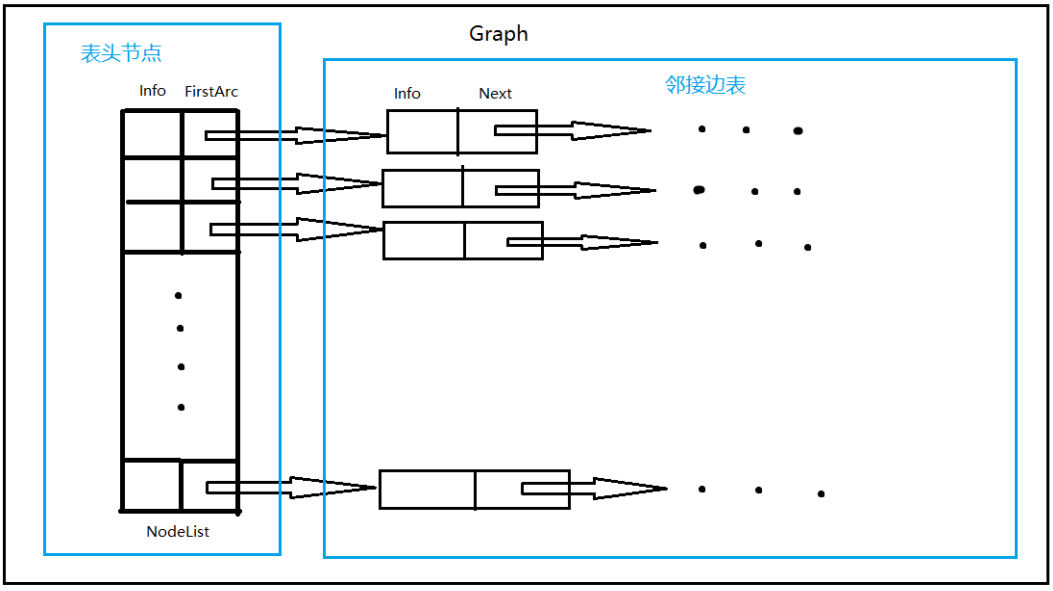

图 3-2 换乘网邻接表结构

3.2.6 路线推荐排行榜

路线推荐排行榜为 3.2.4 中的全部路线经过过滤和筛选排序后前三条路线存储空间，其定义为：

```
62  extern vector<Route> rank_list;
```

3.3 定义和声明

3.3.1 状态值定义

各项状态值定义如表 1-1 所示。

表 1-1 异常参数及其含义对照表

状态值	含义 1	含义 2	含义 3
0	FALSE	ERROR	----
1	TRUE	OK	----
-1	INFEASTABLE	----	----
-2	OVERFLOW	----	----

3.3.2 其他定义

程序的全部定义都卸载 define.h 头文件中，用户可以根据需求方便地进行修改。

```
63  #define MAX_LINE_NUM 10//最大线路数量
64  #define MAX_STN_NUM 100//每条线路最大车站数量
65  #define OPEN_TIME 21600//首班车开始时间
66  #define CLOSE_TIME 82800//末班车时间
67  #define STOP_TIME 60//停车原始时间
68  #define TRAVEL_TIME 120//行车原始时间
69  #define INTERCHANGE_TIME 210//换乘时间
70  typedef int status;
```

四、详细设计

4.1 构图部分

构图部分的目的有如下几个：首先将储存在硬盘上的文件读入到内存中，在这个过程中将其格式化；然后通过内存中读入的数据构建全图邻接表（见 3.2.2），并最终将邻接表以全局变量的形式存储在内存当中。整个过程的流程图如图 4-1 所示。

此外，构图部分还包含必要的测试用函数，用于在调试模式下输出特定的内存中存储信息或函数运算结果，这些函数仅在调试时进行调用。

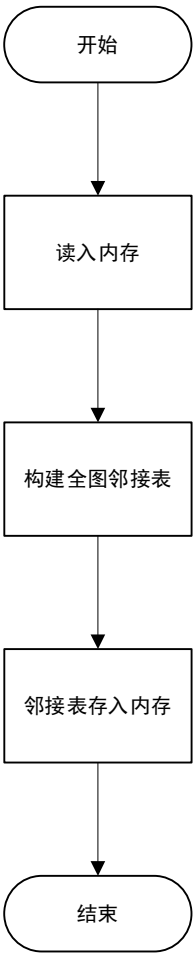

图 4-1 构图模块流程图

4.1.1 文件读取函数

功能： 从文件中将数据读入内存

输入参数： path 数据文件所在路径

返回值： 状态值(int)

返回 INFEASTABLE 表示文件打开失败；

返回 OK 表示文件读取成功。

文件读取函数首先读取.in 格式文件的信息，然后将信息通过遍历存储在内存当中。当前，配置文件的格式并没有做特定的调整，在之后会引入 json 格式的标准化配置文件便于跨平台传输信息。

读取线路信息时程序判断一座车站是否为换乘车站，如果是的话则在数据域中继续存储该车站可换乘的线路信息，知道将一条线路上的全部车站遍历完成为止。

4.1.2 测试输出内存数据函数

功能： 该函数仅用于测试！

将内存中存储的全部信息按照特定的的格式打印出来

输入参数： 无

返回值： 状态值(int)

返回 OK 表示打印成功。

函数在测试时被调用，打印出内存中储存的值，共判断是否出现输入问题。

4.1.3 构建全图邻接表函数

功能：通过内存中的数据构建无向图

输入参数：线路信息静态数组

出口参数：全图邻接表

函数使用了一个指针来遍历每条线路的全部车站，当指针指向某座车站时，可以判断当前车站是否为换乘车站。如果不是换乘车站，则只需将这座车站的前后两站加入这座车站邻接表静态数组结点引出的车站链表当中；如果是换乘车站，则需要将可以换乘的本站其他站台也加入车站链表当中。

最后，还需要注意将链表的尾巴设为NULL，方便检索以及避免出现异常问题。

4.1.4 车站结点返回函数

功能：输入车站编号返回车站在邻接表当中的结点指针。

入口参数：车站编号结构体

出口参数：函数工作状态、该车站在邻接表当中的结点指针

本函数用于在邻接表中快速索引对应车站编号的结构体，一般用于换乘站检索时使用。

4.1.5 打印邻接表函数

功能：仅用于测试，打印内存中的邻接表。

入口参数：内存中保存的邻接表

出口参数：状态值，OK 表示打印成功，ERROR 表示打印错误。

该功能仅在测试模式下进行调用，用于检查邻接表的结构是否正确，有无其他异常等。

4.2 时间运算部分

时间运算部分仅包含多个零散的时间运算相关函数，并定义了多种结构体。其架构图如图 4-2 所示。

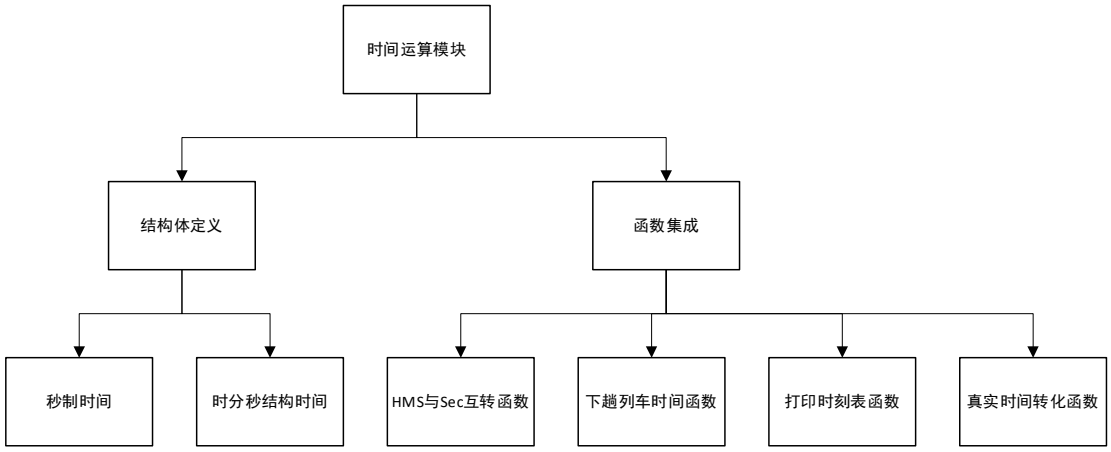

图 4-2 时间运算模块总架构

4.2.1 结构体定义

在本模块中，引入了两个结构体。一个是秒制时间 SecTime，它由一个 10 进制的秒数决定的，由于其较好的连续性，在程序中用于累计时间（秒数）。另一个是习惯时间 HMSTime，它由两个 60 进制的 int 型的秒、分和一个 24 进制的小时组成，由于其更贴近于人们的日常使用习惯，因此在程序中一般将秒制时间转化为 HMS 时间后输出给图形化界面。

两个结构的具体定义详见 3.1 节。

4.2.2 Sec2Time 时转函数

函数名称：Sec2Time

函数功能：将秒制时间转化为 HMS 习惯时间

入口参数：秒制时间 Sec

出口参数：习惯时间结构体 HMSTime

秒数和时间之间的相互转化有如下关系：

$$h = \left\lfloor \frac{n}{3600} \right\rfloor$$

$$m = \left\lfloor \frac{n\%3600}{60} \right\rfloor$$

$$s = n\%60$$

按照如上的方式即可构造秒数 n 到时间 h,m,s 的转换函数。

4.2.3 Time2Sec 时转函数

函数名称：Time2Sec

函数功能：将 HMS 习惯时间转化为秒制时间

入口参数：习惯时间结构体 HMSTime

出口参数：秒制时间 Sec

时间和秒数的互转函数有如下关系：

$$n = h \times 3600 + m \times 60 + s$$

按照如上的方式即可构造时间 h,m,s 到秒数 n 的转换函数。

4.2.4 下趟列车时间计算函数

函数名称：NextTrainTime

函数功能：计算下一趟列车到达时间

入口参数：当前时间结构体

出口参数：下趟列车到达时间结构体

函数首先通过配置文件中设定的规则在内存中生成当前线路当前车站的当前时间附近的时刻表，然后计算当前时间和下趟列车到达时间之间的时间差，分析下趟列车还有几分几秒能够到站。

4.2.5 时刻表打印函数

函数名称：PrintSchedule

函数功能：打印当前车站一整天的时刻表

入口参数：时间规则配置文件数据

出口参数：将时刻表打印在屏幕上

时刻表打印函数可以将本站一整天的全部列车到发时刻输出。

4.2.6 真实时间计算函数

函数名称：CalRealTime

函数功能：计算带有拥挤度影响的真实乘车时间

入口参数：理想乘车时间

出口参数：真实乘车时间

真实时间计算函数可以将纯理想情况下的理想乘车时间转化为带有拥挤度因素影响的真实乘车时间。各站之间的拥挤度指数存储在程序的配置文件当中，除了预先存储的拥挤度指数之外，函数还可以识别人工临时修改的各线路各区间拥挤度。

4.3 票价计算部分

票价计算部分包括多个和票价计算有关的功能函数，整体部分的架构图如图4-3 所示。

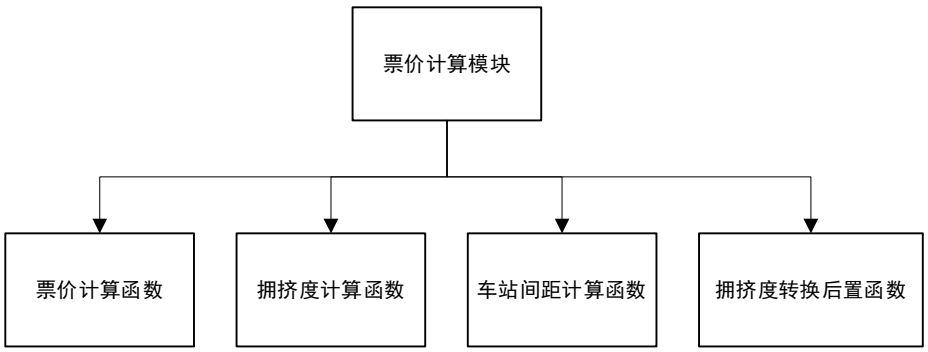

图 4-3 票价计算模块架构图

4.3.1 票价计算函数

函数名称：PriceCal

函数功能：根据地铁票价规则计算票价

入口参数：乘车里程

出口参数：所需票价

票价计算规则如下：

按里程分段计价

4 公里以内（含 4 公里）2 元；

4-12 公里（含 12 公里），1 元/4 公里；

12-24 公里（含 24 公里），1 元/6 公里；

24-40 公里（含 40 公里），1 元/8 公里；

40-50 公里（含 50 公里），1 元/10 公里；

50 公里以上，1 元/20 公里；

这些参数目前内置在程序当中，但将来可以根据需要将其外置到配置文件当中，方便随时修改配置。

4.3.2 拥挤度计算函数

函数名称：CongestionCal

函数功能：查询配置文件当中存储的当前线路当前时间拥挤度

入口参数：当前时间、线路编号

出口参数：当前时间这条线路的拥挤度

为了真实模拟现实中地铁线路的运行情况，程序为每条线路在不同的时间都安排了不同的拥挤度。本函数可以根据当前时间和这条线路的编号，在配置文件中查询当前所需的拥挤度。

拥挤度预设如下：

• 上下班类型：1 号线，3 号线；

7:30-9:00，拥挤度 80%；16:30-18:30，拥挤度 75%；其余时间拥挤度 40%；

• 购物类型：8 号线；

9:30-15:00，拥挤度 65%；其余时间拥挤度 20%；

• 娱乐类型：6 号线，7 号线；

19:00-22:00，拥挤度 65%；其余时间拥挤度 15%；

• 城际交通类型：2 号线，4 号线；

全天拥挤度 50%；

4.3.3 里程计算函数

函数名称：StationDist

函数功能：计算各线路每两站之间的里程

入口参数：线路信息配置文件

出口参数：两站间历程

由于目前程序采用的是简化设定，即将线路总长度平均到每两站之间来粗略地估计两站之间的站间距，还暂时不需要一个单独的函数来完成这样的工作。但是为了后期的继续升级，还是留了一个函数用来计算某条线路某两站之间的里程。

4.3.4 人流量判断函数

函数名称：TrafficJudge

函数功能：判断拥挤度对应的人流量

入口参数：拥挤度

出口参数：人流量分类（0=宽松，1=一般，2=拥挤）

由于拥挤程度难以直接对应到时间的影响之上，因此需要用一个函数来转化拥挤度到时间指数之上。

对应关系如下：（设c为拥挤度）

1. 宽松（flag==0）：

$$0 \leq c \leq 0.2$$

2. 一般（flag==1）：

$$0.2 < c \leq 0.62$$

3. 拥挤（flag==2）：

$$0.62 < c \leq 1$$

在数学上，本函数可以表示为一个由连续到离散的函数来表示，其定义如下：

在此处键入公式。

4.4 路线规划部分

路线规划部分是本程序的主要部分，其架构如图 4-4 所示。

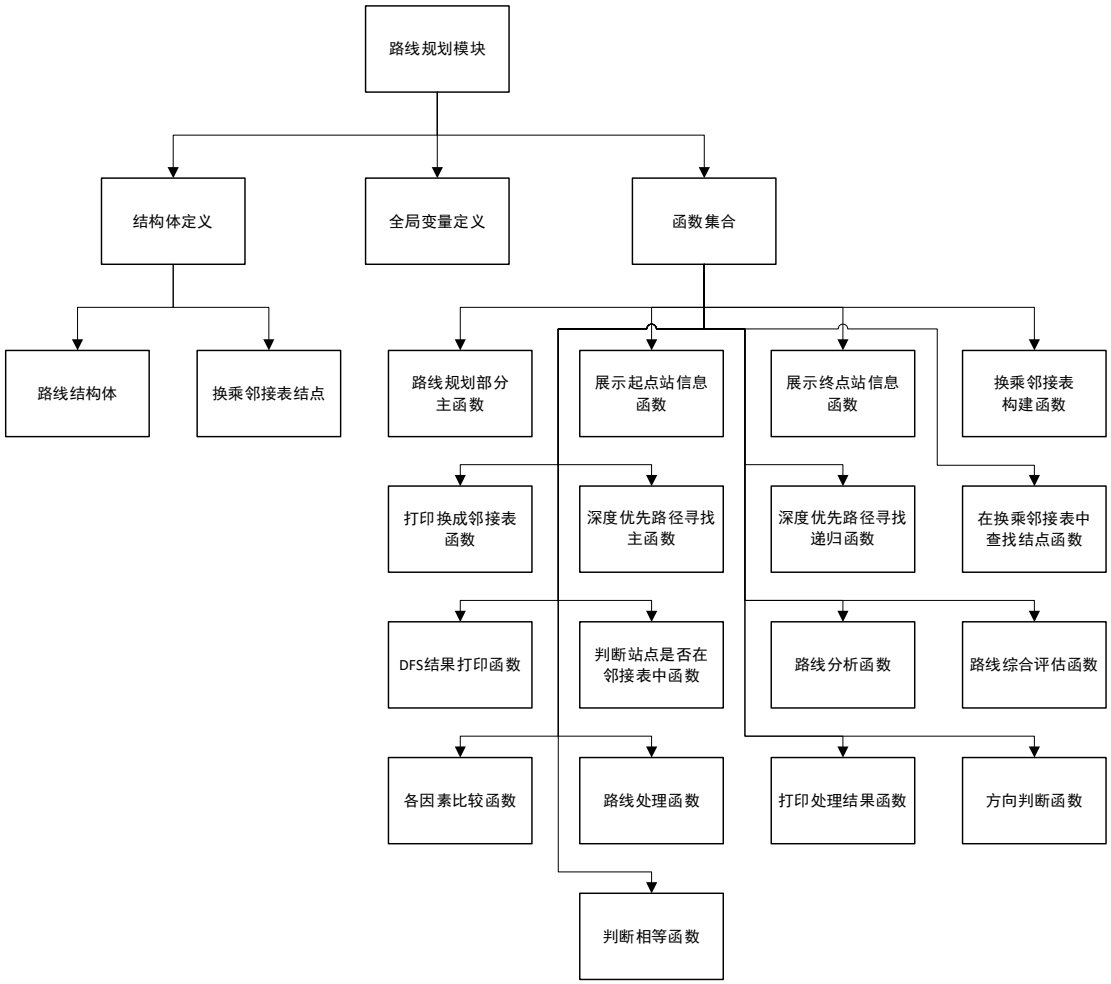

图 4-4 路径规划部分架构

可以看到，在这个模块当中包含不少的功能函数，为了保证模块接口的集成性，在路径规划模块中，我设置了一个路径规划模块的主函数，通过这个主函数进行模块的输入输出和模块中众多功能函数的调用功能。因此，在此画出此主函数的流程图，也就代表了这个路径规划模块的流程图。流程图具体如图4-5所示。

流程图中灰色代表在正常使用中不会调用的函数部分，但是在调试过程中会被调用方便查错。

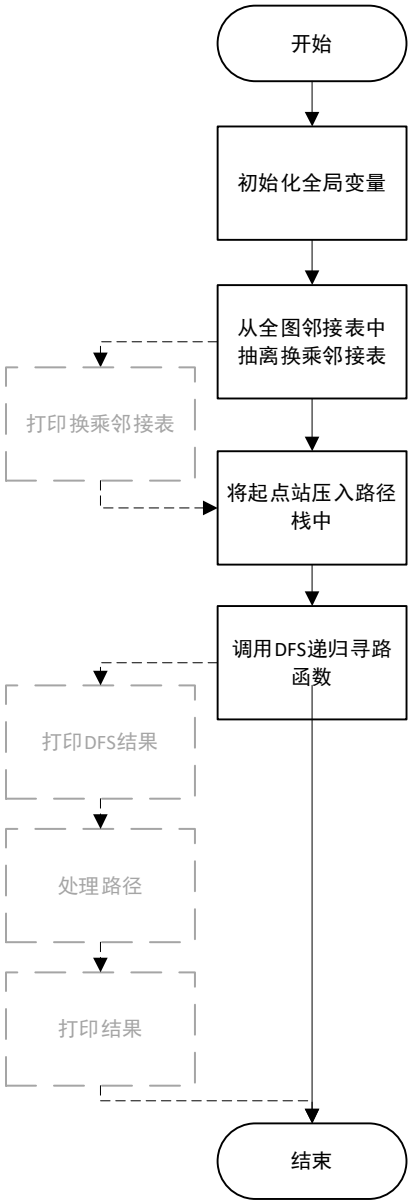

图 4-5 路径规划模块流程图

4.4.1 路径规划主函数

路径规划主函数具体定义在上文中已经说明，在此不再赘述。

4.4.2 打印始末车站信息

函数名称：ShowStartStation

函数功能：打印用户选定的始末车站的信息

入口参数：用户选择的始末车站

出口参数：屏幕上输出始末车站信息

本函数将在用户选定始末车站后在交互框中输出提示信息，具体如图 4-6 所示。

图 4-6 函数输出的信息

从图中，我们可以看到在右侧的交互框中，程序输出了线路信息、车站信息列表及当前车站、是否为换乘站以及下趟列车的到达和发车时间。具体如下：

```
71  武汉地铁 2 号线
72  车站数量:38 座;      线路全长:60.8km;      最大载客量:2590 人。
73  201 天河机场站
74  202 航空总部站
75  203 宋家岗站
76  204 巨龙大道站  可换乘 7 号线,
77  205 盘龙城站
78  206 宏图大道站  可换乘 3 号线,8 号线,
79  207 常青城站
80  208 金银潭站
81  209 常青花园站  可换乘 6 号线,
82  210 长港路站
83  211 汉口火车站
84  212 范湖站  可换乘 3 号线,
85  213 王家墩东站  可换乘 7 号线,
86  214 青年路站
87  215 中山公园站
88  216 循礼门站  可换乘 1 号线,
89  217 江汉路站  可换乘 6 号线,
90  218 积玉桥站
91  219 螃蟹岬站  可换乘 7 号线,
92  220 小龟山站
93  221 洪山广场站  可换乘 4 号线,
94  222 中南路站  可换乘 4 号线,
95  223 宝通寺站
96  224 街道口站  可换乘 8 号线,
```

```
225 广埠屯站
226 虎泉站
227 杨家湾站
228 光谷广场站
229 珞雄路站
230 华中科技大学站
231 光谷大道站
232 佳园路站
233 光谷火车站
234 黄龙山路站
235 金融港北站
236 秀湖站
237 藏龙东街站
238 佛祖岭站
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
您的起始站点是：华中科技大学站
本站不是换乘站.
下趟列车将在 1 分 25 秒后到达，将在 2 分 25 秒后离站.
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

此外，还会在程序的最上方的时间显示框中显示下趟列车到达的剩余时间，尽管在右侧的输出框中已经输出了下趟列车的到达时间，但是出于程序人性化设计及用户的立场考虑，还是在更加醒目的位置添加了此信息。

4.4.3 抽离换乘邻接表函数

函数名称：BuildTransferGraph

函数功能：将换乘车站和用户选定的始末车站从全图邻接表当中抽离出来作为一个生成子图，简化计算降低时间利用

入口参数：全图邻接表

出口参数：抽离出来的换乘车站邻接表

由于地铁网络换乘站占比较低，因此考虑将非换乘车站从邻接表表示的无向图当中去掉，以此来简化程序数据量和运算开销。而两座换乘车站中间的原有一般车站，则将化作换乘车站之间边上的边权处理。

本函数流程图如图 4-7 所示。

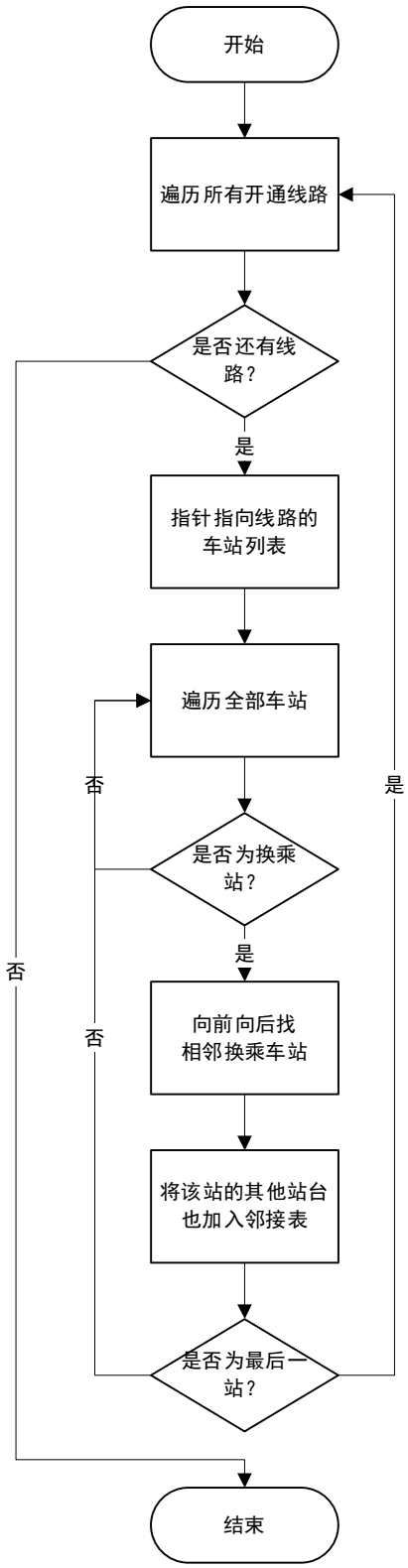
开始
遍历所有开通线路
是否还有线路？
是
指针指向线路的车站列表
遍历全部车站
否
是
是否为换乘站？
是
否
向前向后找相邻换乘车站
否
将该站的其他站台也加入邻接表
是否为最后一站？
结束

图 4-7 抽离函数流程图

4.4.4 DFS 迭代函数

函数名称：DFS

函数功能：递归进行深度搜索并存储每次搜索结果

入口参数：当前路径栈、目的地车站、当前深度、最大搜索深度

出口参数：一组可行路线的原始数据（未经排序但每条路线的各属性均已完成）

DFS 的搜索递归流程图如图 4-8 所示。

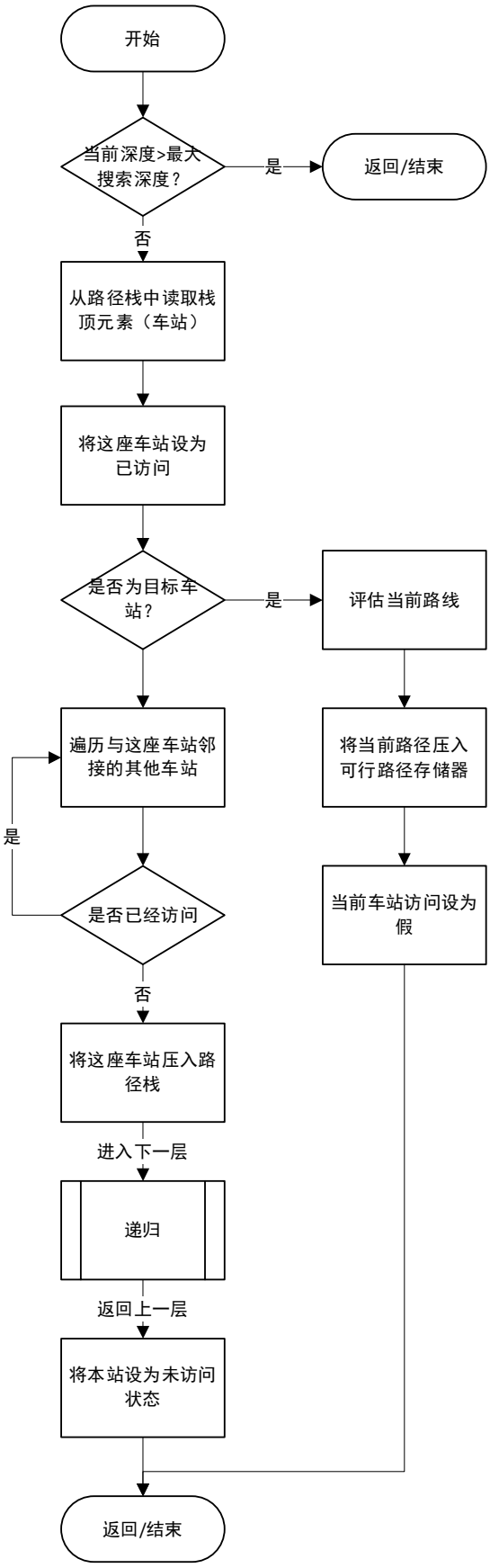

图 4-8 深度优先搜索流程图

4.4.5 路径评估函数

函数名称：RouteEvaluation

函数功能：对 DFS 搜索得到的每一条路径进行评估

入口参数：需要评估的路线（路径各项属性为空）

出口参数：完成评估后的路径（路径各项属性已经完善）

在每次 DFS 找到一条可行路径后，都会调用本函数对刚刚存入的路径进行评估。在评估时，函数会调用其他功能函数来分析路径全长、乘车所需真实时间、换乘次数、票价、整体拥挤度和路径加权推荐分数等信息，并保存在路径结构体的数据域当中。

4.4.6 路径打印函数

函数名称：PrintResult

函数功能：将排序后的前三条路径打印在输出框中

入口参数：排名后的 rank_list

出口参数：将结果打印在显示屏上

路径打印实例如下：

```
117 ======================路线推荐======================
118 出发时间:  15:25
119 华中科技大学站 -------> 横店站
120 以综合排序最优方式推荐路线
121
122 推荐路线 1:
123 用时: 01 时 10 分 30 秒;
124 票价: 9 元;
125 换乘次数: 3 次;
126 总里程: 42.8042 公里;
127 宽松区间: 20 条, 一般区间: 8 条, 拥挤区间: 0 条.
128 总体拥挤指数: 0.057
129 预计到达时间: 16:36
130 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->街道口站--2 号线)-{换乘 8 号线}-(8 号线---->街道口站-->小洪山站-->洪山路站-->水果湖站-->中南医院站-->岳家嘴站-->赵家条站-->竹叶山站-->宏图大道站--8 号线)-{换乘 2 号线}-(2 号线---->巨龙大道站--2 号线)-{换乘 7 号线}-(7 号线---->巨龙大道站-->腾龙大道站-->横店站--7 号线)-终点
131
132 推荐路线 2:
133 用时: 01 时 14 分 18 秒;
134 票价: 10 元;
135 换乘次数: 1 次;
136 总里程: 52.173 公里;
137 宽松区间: 11 条, 一般区间: 17 条, 拥挤区间: 0 条.
138 总体拥挤指数: 0.121
139 预计到达时间: 16:39
140 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->循礼门站-->王家墩东站--2 号线)-{换乘 7 号线}-(7 号线---->武汉商务区站-->园博园北站-->巨龙大道站-->腾龙大道站-->横店站--7 号线)-终点
```

```
141
142 推荐路线 3:
143 用时: 01 时 10 分 00 秒;
144 票价: 10 元;
145 换乘次数: 2 次;
146 总里程: 56.731 公里;
147 宽松区间: 25 条, 一般区间: 6 条, 拥挤区间: 0 条.
148 总体拥挤指数: 0.039
149 预计到达时间: 16:35
150 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->街道口
    站--2 号线)-{换乘 8 号线}-(8 号线---->街道口站-->小洪山站-->洪山路站-->水果
    湖站-->中南医院站-->岳家嘴站-->徐家棚站--8 号线)-{换乘 7 号线}-(7 号线---->
    武汉商务区站-->园博园北站-->巨龙大道站-->腾龙大道站-->横店站--7 号线)-终点
```

4.5 界面交互部分

4.5.1 有关 Qt

界面交互功能及可视化界面采用 Qt 实现。Qt 是一个小部件工具包，用于创建图形用户界面以及跨平台 应用上的各种软件和硬件平台，例如运行的 Linux，视窗，MACOS，Android 的或嵌入式系统，其基础代码库几乎没有变化，而仍然是具有本机功能和速度的本机应用程序。Qt 用于开发在所有主要桌面平台以及大多数移动或嵌入式平台上运行的图形用户界面（GUI）和多平台应用程序。使用 Qt 创建的大多数 GUI 程序都具有本机外观的界面，在这种情况下，Qt 被归类为小部件工具箱。还可以开发非 GUI 程序，例如服务器的命令行工具和控制台。使用 Qt 的此类非 GUI 程序的示例是 Cutelyst Web 框架。Qt 支持各种编译器，包括 GCC C ++编译器，Visual Studio 套件，通过 PHP5 扩展的 PHP，并具有广泛的国际化支持。Qt 还提供了 Qt Quick，其中包括一种称为 QML 的声明性 脚本语言，该语言允许使用 JavaScript 提供逻辑。借助 Qt Quick，可以实现针对移动设备的快速应用程序开发，同时仍然可以使用本机代码编写逻辑，以实现最佳性能。其他功能包括 SQL 数据库访问，XML 解析，JSON 解析，线程管理和网络支持。

4.5.2 信号和槽

信号与槽（Signal & Slot）是 Qt 编程的基础，也是 Qt 的一大创新。因为有了信号与槽的编程机制，在 Qt 中处理界面各个组件的交互操作时变得更加直观和简单。

信号（Signal）就是在特定情况下被发射的事件，例如 PushButton 最常见的

信号就是鼠标单击时发射的 clicked() 信号，一个 ComboBox 最常见的信号是选择的列表项变化时发射的 CurrentIndexChanged() 信号。

GUI 程序设计的主要内容就是对界面上各组件的信号的响应，只需要知道什么情况下发射哪些信号，合理地去响应和处理这些信号就可以了。

槽（Slot）就是对信号响应的函数。槽就是一个函数，与一般的 C++函数是一样的，可以定义在类的任何部分（public、private 或 protected），可以具有任何参数，也可以被直接调用。槽函数与一般的函数不同的是：槽函数可以与一个信号关联，当信号被发射时，关联的槽函数被自动执行。

信号与槽关联是用 QObject::connect() 函数实现的，其基本格式是：

```
151 QObject::connect(sender, SIGNAL(signal()), receiver, SLOT(slot()));
```

4.5.3 图形化界面展示

本程序的图形化界面如图 4-9 所示。

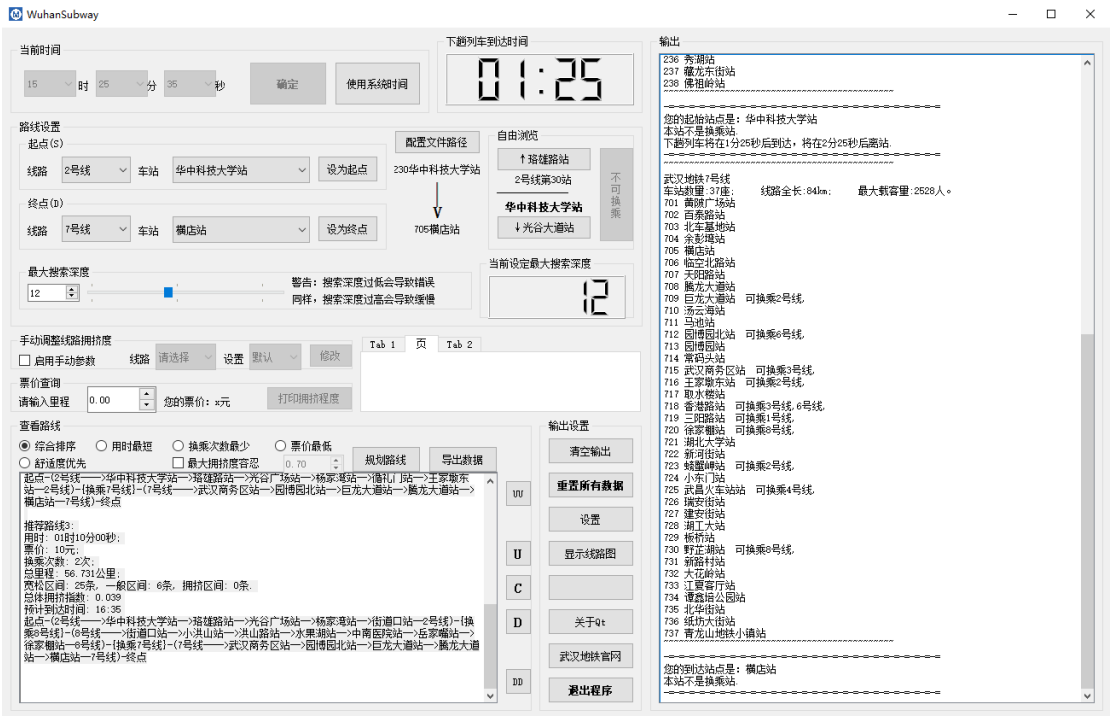

图 4-9 图形化界面展示

程序的图标如图 4-10 所示。

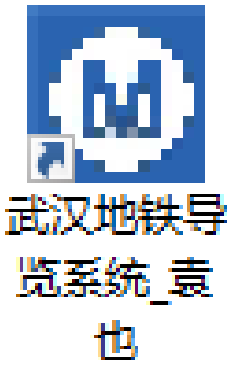

图 4-10 程序图标

五、系统实现

5.1 开发环境

Microsoft Visual Studio 2019 IDE 开发核心

Qt Creator 5.14.2 开发交互界面

编译器版本：MinGW 7.3.0 32-bit for C++ / MinGW 7.3.0 64-bit for C++

Qmake 版本：QMake version 3.1

系统：Windows 10 x64 Home

C++标准：C++ 11

文字编码格式：GB2312

支持包：C++ Standard Template Library、Qt 库（提供界面支持）

5.2 主要函数模块

5.2.1 主程序模块

主程序模块是程序的入口。

调用关系：调用类 WuhanSubway（界面）

声明：int main(int argc, char *argv[])

入口参数：命令行中传入的参数

出口参数：状态值

在程序中首先将定义的类 WuHanSubway 实例化，生成变量 w，然后调用 w 的函数 show()。

5.2.2 交互界面模块

交互界面模块提供窗口的显示，组建的信号指令传输，其他的输入输出以及错误信息的抛出和报错等功能，是程序向用户开放的直接入口和对话通道。交互界面模块中的函数基本都是程序窗口组件的槽函数。

1. 声明：WuhanSubway::WuhanSubway(QWidget *parent) : QWidget(parent),

ui(new Ui::WuhanSubway);

调用关系：全局变量 path, depth_m; 调用函数 input(), BuildMap()

入口参数：无

出口参数：界面窗口

2. 声明：void on_Time_OK_clicked();

调用关系：全局变量_h, _m, _s, tt (TimeHMS 结构体)

入口参数：无

出口参数：无

3. 声明：void on_Time_hh_activated(int index);

调用关系：无

入口参数：选中下标

出口参数：无

4. 声明：void on_Time_mm_activated(int index);

调用关系：无

入口参数：选中下标

出口参数：无

5. 声明：void on_Time_ss_activated(int index);

调用关系：无

入口参数：选中下标

出口参数：无

6. 声明：void on_Sline_activated(int index);

调用关系：全局变量 line.

入口参数：选中下标

出口参数：无

7. 声明：void on_emptyop_clicked();
   调用关系：无
   入口参数：无
   出口参数：无

8. 声明：void on_CloseWindow_clicked();
   调用关系：无
   入口参数：无
   出口参数：无

9. 声明：void on_Sstation_activated(int index);
   调用关系：无
   入口参数：选中下标
   出口参数：无

10. 声明：void on_SetAsStartP_clicked();
    调用关系：全局变量 lineid_s, stid_s, lineid_d, stid_d, line; 调用函数 ShowStartStation()
    入口参数：无
    出口参数：无

11. 声明：void on_Dline_activated(int index);
    调用关系：全局变量 lineid_s, stid_s, lineid_d, stid_d, line; 调用函数 ShowDestStation()
    入口参数：选中下标
    出口参数：无

12. 声明：void on_Dstation_activated(int index);
    调用关系：无
    入口参数：选中下标
    出口参数：无

13. 声明：void on_SetAsDestP_clicked();
    调用关系：无
    入口参数：无
    出口参数：无

14. 声明：void on_ShowMap_clicked();
    调用关系：无
    入口参数：无
    出口参数：无

15. 声明：void on_UseSystemTimeBtn_clicked();
    调用关系：无
    入口参数：无
    出口参数：无

16. 声明：void on_ResetAll_clicked();
    调用关系：无
    入口参数：无
    出口参数：无

17. 声明：status ShowStartStation(TimeHMS now, int Sline, int SID);
    调用关系：无
    入口参数：HMS 时间结构体、起点站线路、起点站编号
    出口参数：状态值

18. 声明：void on_MDepthSlider_actionTriggered(int action);
    调用关系：无
    入口参数：动作幅度
    出口参数：无

19. 声明：void on_MDepthBox_valueChanged(int arg1);

    调用关系：无

    入口参数：调节的参数

    出口参数：无

20. 声明：status ShowDestStation(int Dline, int DID);

    调用关系：无

    入口参数：终点站线路、终点站编号

    出口参数：状态值

21. 声明：void on_AboutQT_clicked();

    调用关系：无

    入口参数：无

    出口参数：无

22. 声明：void on_MDepthSlider_valueChanged(int value);

    调用关系：无

    入口参数：输入的数据

    出口参数：无

23. 声明：void on_R1_clicked();

    调用关系：无

    入口参数：无

    出口参数：无

24. 声明：void on_R2_clicked();

    调用关系：无

    入口参数：无

    出口参数：无

25. 声明：void on_R3_clicked();

调用关系：无

入口参数：无

出口参数：无

26. 声明：void on_PlanRoute_clicked();

调用关系：无

入口参数：无

出口参数：无

27. 声明：void on_CButn_clicked();

调用关系：无

入口参数：无

出口参数：无

28. 声明：void on_Outp_ret_clicked();

调用关系：无

入口参数：无

出口参数：无

29. 声明：void on_SetPathBtn_clicked();

调用关系：无

入口参数：无

出口参数：无

30. 声明：void on_DDButn_clicked();

调用关系：无

入口参数：无

出口参数：无

31. 声明：void on_Website_butn_clicked();

调用关系：无

入口参数：无

出口参数：无

32. 声明：void on_PrevStation_clicked();

调用关系：无

入口参数：无

出口参数：无

33. 声明：void on_NextStation_clicked();

调用关系：无

入口参数：无

出口参数：无

34. 声明：void on_TransferBtn_clicked();

调用关系：无

入口参数：无

出口参数：无

35. 声明：void on_DistBox_valueChanged(double arg1);

调用关系：无

入口参数：设定的 double 型参数

出口参数：无

36. 声明：void on_isManualBox_stateChanged(int arg1);

调用关系：无

入口参数：设定的 int 型参数

出口参数：无

37. 声明：void on_R6_stateChanged(int arg1);

调用关系：无

入口参数：设定的 int 型参数

出口参数：无

38. 声明：void on_R4_clicked();

调用关系：无

入口参数：无

出口参数：无

39. 声明：void on_R5_clicked();

调用关系：无

入口参数：无

出口参数：无

40. 声明：void on_TrafficYesBtn_clicked();

调用关系：无

入口参数：无

出口参数：无

41. 声明：void on_Traffic_Line_Set_activated(int index);

调用关系：无

入口参数：设定的 int 型参数

出口参数：无

42. 声明：void on_PrtTrafBtn_clicked();

调用关系：无

入口参数：无

出口参数：无

以上均为窗口组件的槽函数。

5.2.3 路径规划模块

路径规划模块是程序的主体，提供了主要的规划路径的功能。函数的入口参数和出口参数信息详见 4.4 节，由于内容相同，在此不再赘述。

1. extern status route_main(TimeHMS now, int Sline, int SID, int Dline, int DID, int max_search_depth);
   调用关系：Time2Sec(), BuildTransferGraph(), PrintTransferAdjList(), FindStation(), DFS(), PrintDFSResult(), ProcessingRoute(), PrintResult()

2. extern status ShowStartStation(TimeHMS now, int Sline, int SID);
   调用关系：无

3. extern status ShowDestStation(int Dline, int DID);
   调用关系：无

4. extern status BuildTransferGraph(int Sline, int SID, int Dline, int DID);
   调用关系：无

5. extern status PrintTransferAdjList(void);
   调用关系：无

6. extern void DFS(Route nowR, StNum Destination, int depth, int max_depth);
   调用关系：DFS(), RouteEvaluation(), PrintDFSResult(), FindTransferInAdjList(), isInArray()

7. extern vector<TransferNode>::iterator FindTransferInAdjList(int lineid, int stid);
   调用关系：无

8. extern status PrintDFSResult(void);
   调用关系：无

9. extern bool isInArray(Route nowR, StNum s);
   调用关系：无

10. extern status RouteEvaluation(Route& nowR);
    调用关系：FindTransferInAdjList(), TrafficJudge(), CalRealTime(), PriceCal(), ComprehensiveEvaluation()

11. extern double ComprehensiveEvaluation(Route nowR);
    调用关系：无

12. extern bool comprehensive_cmp(Route a, Route b);
    调用关系：无

13. extern bool time_cmp(Route a, Route b);
    调用关系：无

14. extern bool transfer_cmp(Route a, Route b);
    调用关系：无

15. extern bool price_cmp(Route a, Route b);
    调用关系：无

16. extern bool traffic_cmp(Route a, Route b);
    调用关系：无

17. extern status ProcessingRoute(void);
    调用关系：comprehensive_cmp(), time_cmp(), transfer_cmp()

18. extern status PrintResult(void);
    调用关系：Sec2Time(), JudgeDirection()

19. extern int JudgeDirection(int a, int b);
    调用关系：无

20. extern bool isTrafficEqual(double a, double b);
    调用关系：无

5.2.4 构图模块

构图模块的各函数入口参数及出口参数详见 4.1 节，由于内容重复，在此不再赘述。

1. extern status input(const char* path);
   调用关系：底层文件读取函数

2. extern status Test_Output(void);
   调用关系：无

3. extern StNode* CreateStationNode(Station s);
   调用关系：无

4. extern status BuildMap(void);
   调用关系：CreateStationNode()

5. extern status PrintAdjList(void);
   调用关系：无

6. extern vector<Station>::iterator FindStation(int lineid, int stid);
   调用关系：无

六、程序安装及使用说明

6.1 程序安装

用户首先需要从互联网或其他渠道获取本软件的安装包，名为“wuhan_subway_v1.0_by_YuanYe_setup.exe”。然后可以双击安装包进入安装程序，双击后如图 6-1 所示。

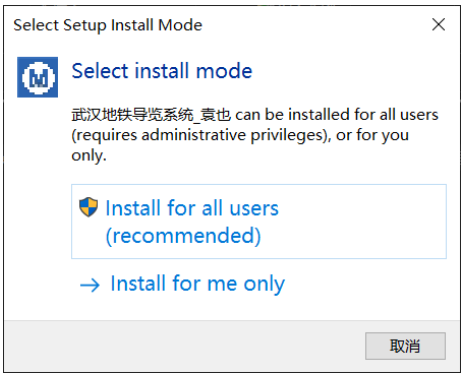

图 6-1 安装模式选择

这时，如果需要为所有用户安装此程序，则选择第一个选项；如果只想自己安装，则选择第二个选项。用户可以根据自己的需求进行选择。

然后进入图 6-2 所示界面，阅读程序使用条例并同意后点击 next 进入下一步。

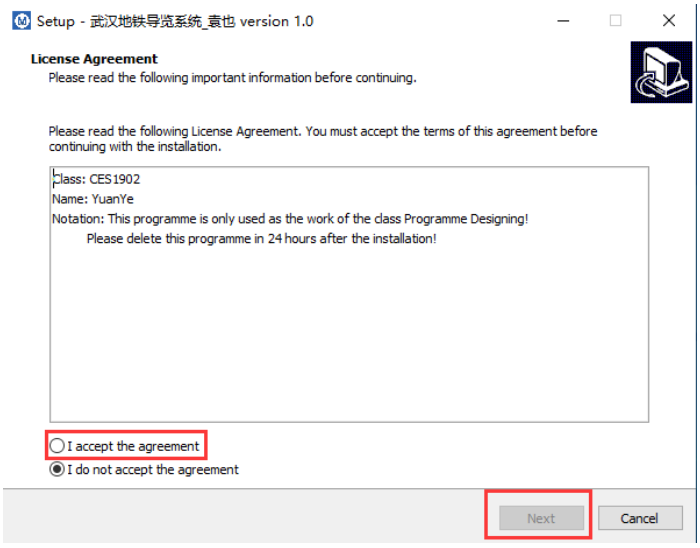

图 6-2 按照红框指示进行操作

点击下一步后，进入如图 6-3 所示的界面，输入安装密码

```
152 U201911808
```

后继续安装，点击下一步，如图 6-4 所示。

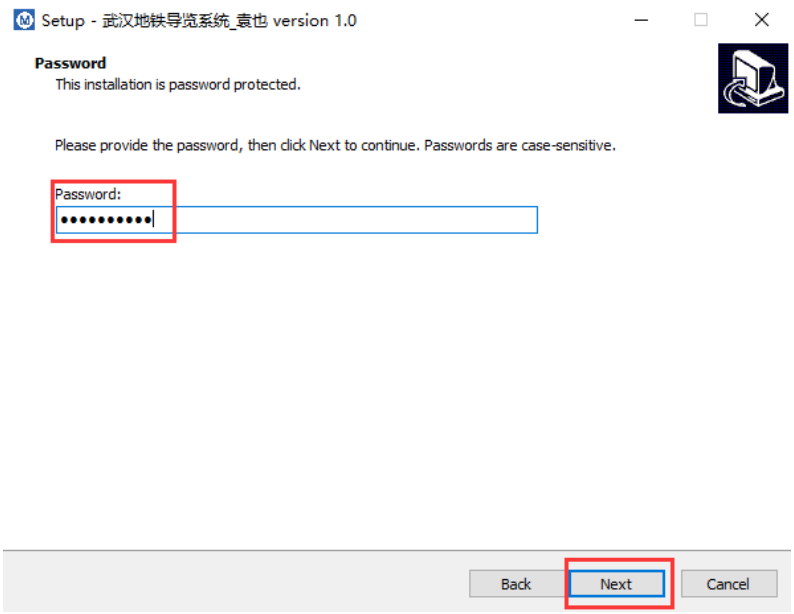

图 6-4 按照红框指示进行操作

之后一直点击下一步，直到显示出如图 6-5 所示窗口后，点击 Finish 完成安装并自动运行程序。

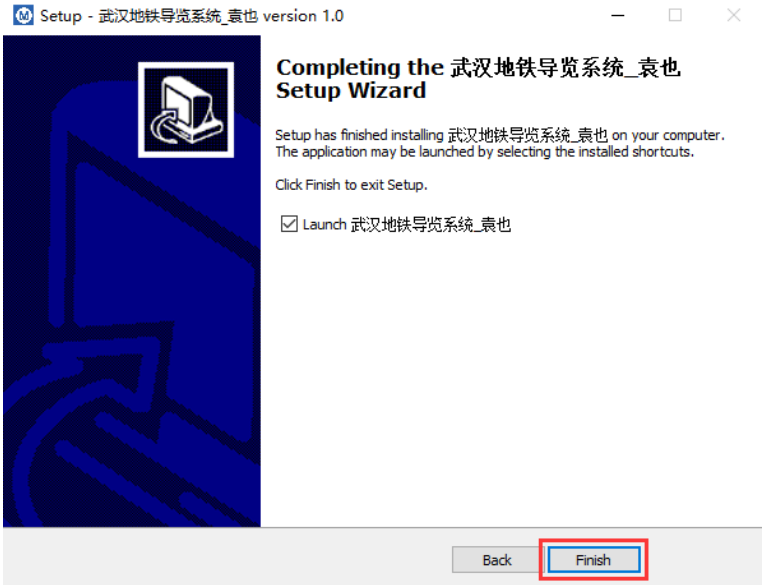

图 6-5 按照红框指示进行操作

6.2 程序使用教程

进入程序后，首先看到如图 6-6 所示的界面。

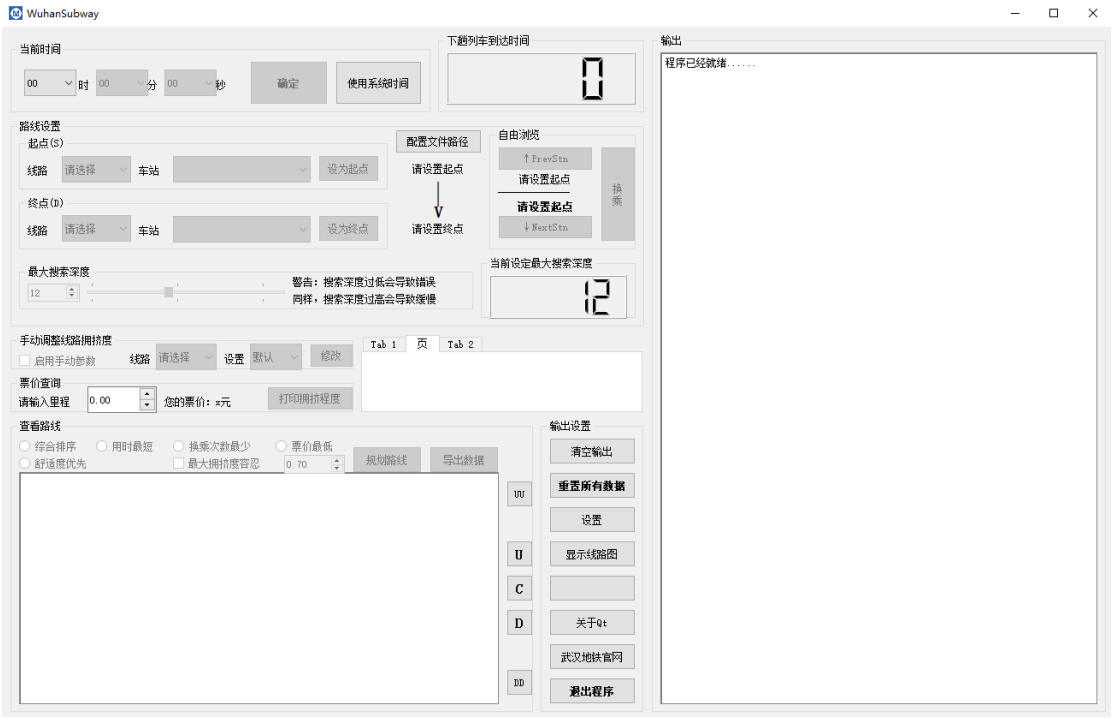

图 6-6 程序主界面

为了提高程序鲁棒性，避免用户因不了解程序数据输入顺序从而导致异常的产生，程序设定了按照规则来引导用户完成各项内容的设置。当前不可填写的信息为灰色框显示，用户无法点击或输入数据；其余则为用户当前可以改变的数据。

6.2.1 查询票价

用户可以在票价查询处输入一定的里程查询对应的票价，如图 6-7 红框处所示。

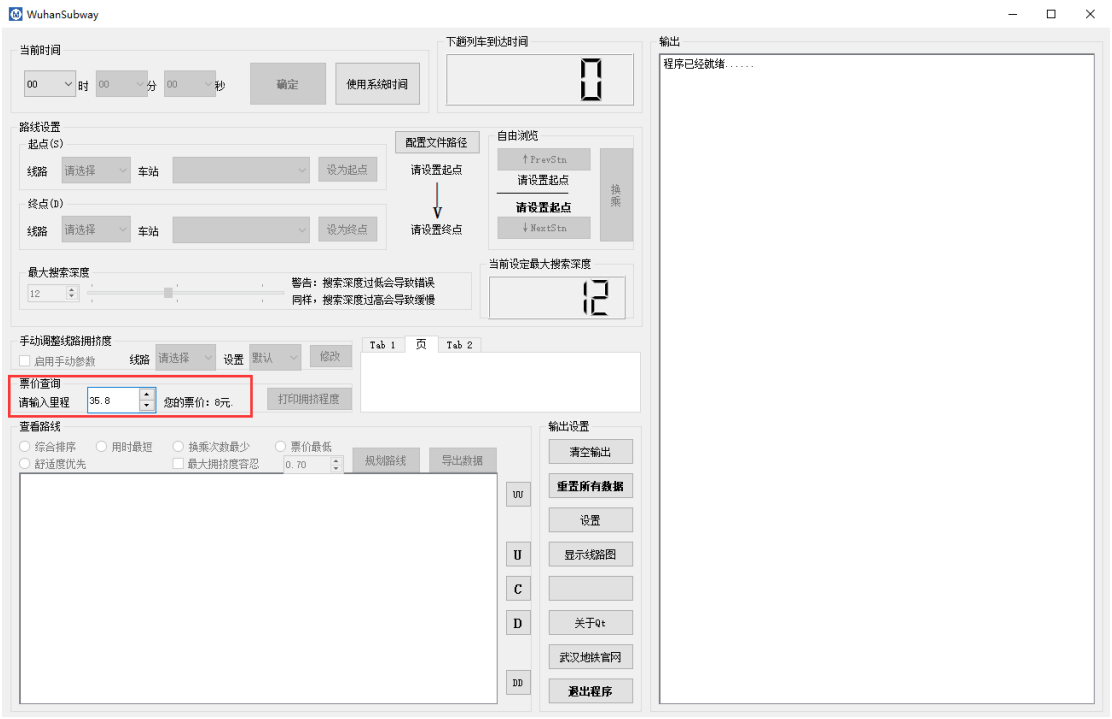

图 6-7 手动查询票价

当然，假如用户并不知道自己的里程，不用担心，在后面的路径规划功能中自带了票价显示功能。

6.2.2 设置当前时间

由于程序可以满足在任何时刻生成路线的功能，因此程序界面上设置了可供用户自由选择出发时刻的菜单栏。如图 6-8 红框所示，按照灰色框的提示即可设置当前时间。此外，用户还可以直接采用系统当前时间，来模拟真实乘车过程中路线查询的功能。

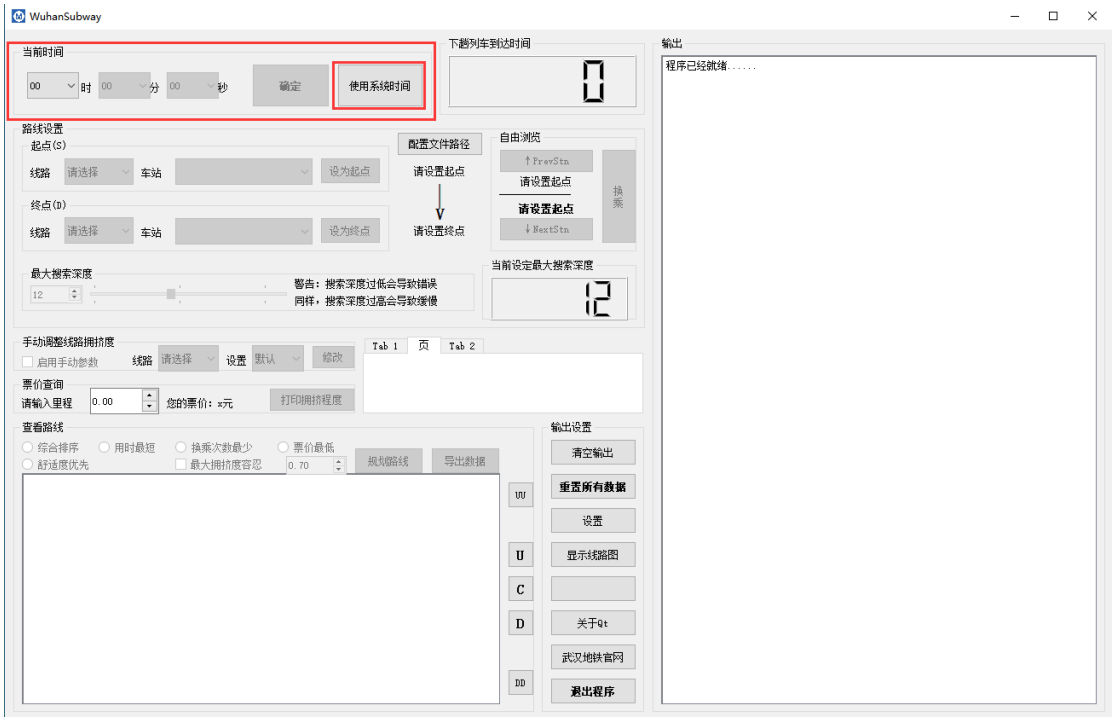

图 6-8 设置时间菜单栏

时间设置完成后，时间设置菜单栏会自动锁定，信息输出框会输出当前时间的文字，如图 6-9 所示。如果此时还想重新设置时间，则需要点击红框中的“重置所有数据进行重设。

图 6-9 设置完成时间后

6.2.3 设置起点站和终点站

按照指示可以选择起点站和终点站，选定一条线路后，车站列表会自动切换到当前线路的车站列表，如图 6-10 所示。选择某条线路时，输出框会打印有关这条地铁线路车站列表的信息。

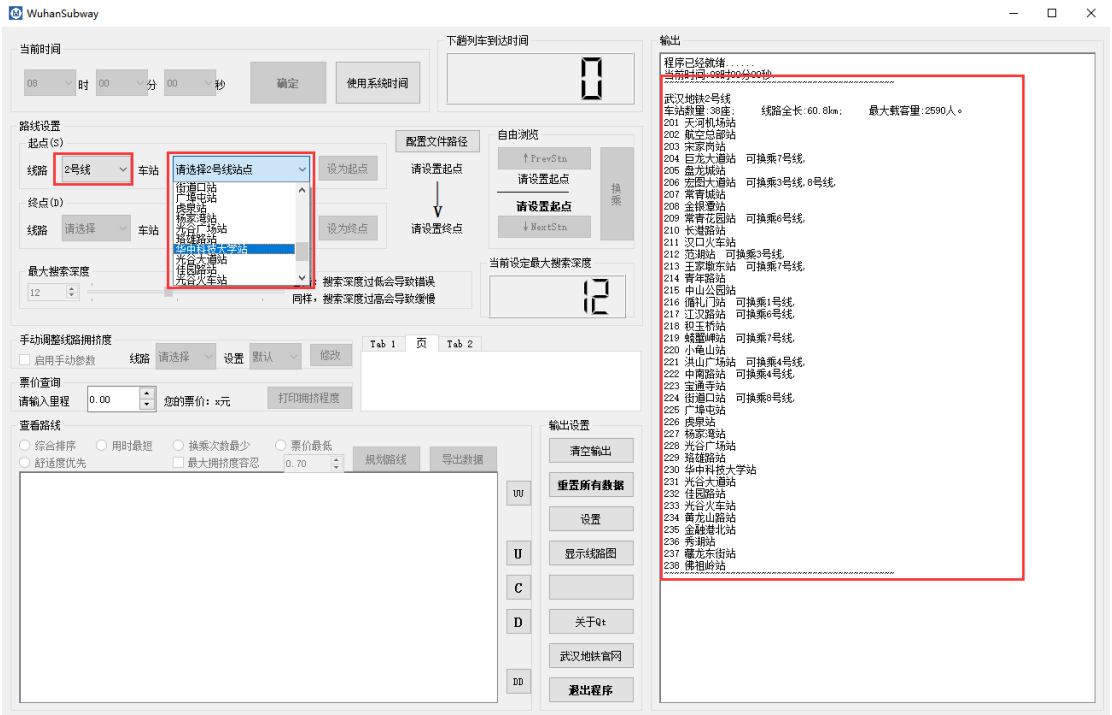

图 6-10 车站选择状态

设置完成后，点击设为起点/设为终点确认数据，起点站终点站设置完成后界面如图 6-11 所示。

图 6-11 路线设置完成后状态

6.2.4 路线自由浏览

可以在如图 6-12 的红色框中进行路线自由浏览，自由选择当前车站的前一站或后一站前往，或者在换乘站换乘其他线路。

图 6-12 自由浏览路线

6.2.5 设置最大搜索深度

可以通过调节数字框或滑动条的方式来调节最大搜索深度。最大搜索深度和线路最大换乘次数成正比，如果搜索深度过低可能会找不到路线，过高则会导致运算缓慢。设置位置如图 6-13 所示，默认为 12.

图 6-13 最大搜索深度设置

6.2.6 手动调整线路拥挤度

程序为了模拟各种突发情况及更加真实的地铁线路情况，设置了手动调整线路拥挤度的功能，默认不选。可以勾选复选框，并调整某条线路的拥挤度，如图 6-14.

图 6-14 手动设置拥挤度菜单栏

6.2.7 路线规划

首先，在图 6-15 所示的 RadioBox 中任选一个路线推荐优先顺序的选项，当然任意选项都可以配合最大拥挤度容忍来使用，之后可以点击规划路线即可生成当前模式下的三条最佳路线。点击导出数据还能导出路线信息分享给自己的好友，自动生成的文件如图 6-16 所示。

图 6-15 路线规划结果

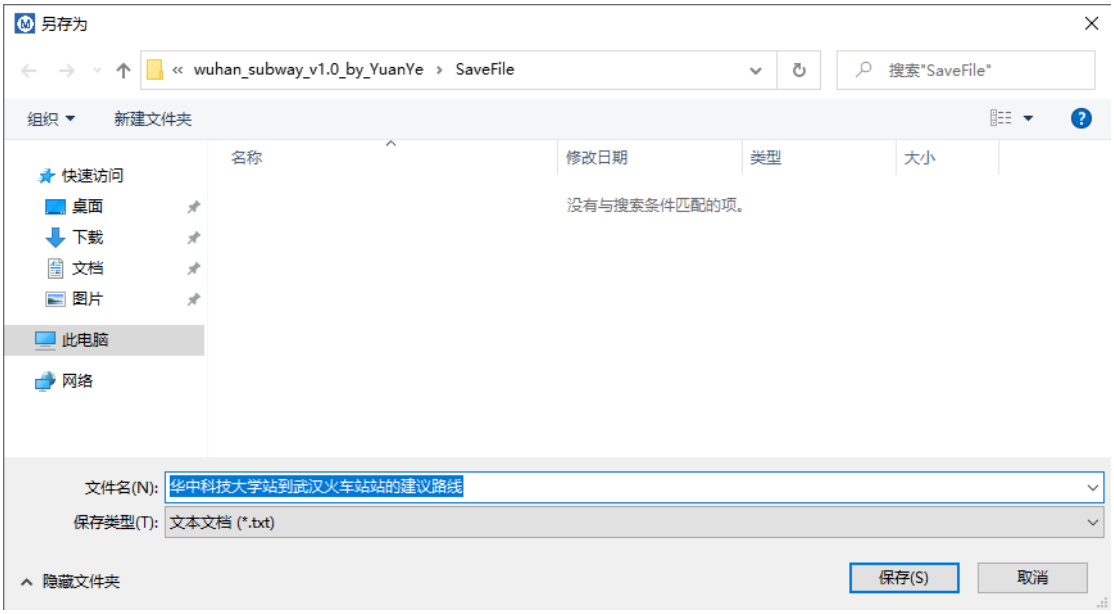

图 6-16 自动生成文件

6.3 程序卸载

在安装路径下找到“unins000.exe”双击运行即可出现下图所示卸载界面，按照指示进行卸载即可。

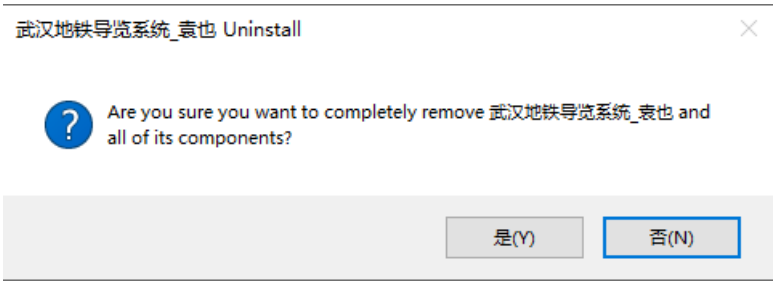

图 6-17 卸载软件

七、运行测试与结果分析

7.1 测试计划

经过严密计划，本次测试将测试程序的以下功能：

1. 选定线路时，展示线路的信息；
2. 选定始末站时，显示站点的信息；
3. 输入里程，计算对应的票价；
4. 选定起点，自由导航前往任意站点；
5. 查看最近一趟车的到站和发车时间；
6. 选定起点终点，选择特定的需求，输出按偏好排序好的路线以供查询；
7. 修改某条线路拥挤度，查看推荐路线的变化。

7.2 测试过程

7.2.1 显示线路信息

当前时间选定系统时间，在起点处设置线路为 2 号线之后，在程序主界面右侧出现关于武汉地铁 2 号线的信息，如图 7-1 红框所示。

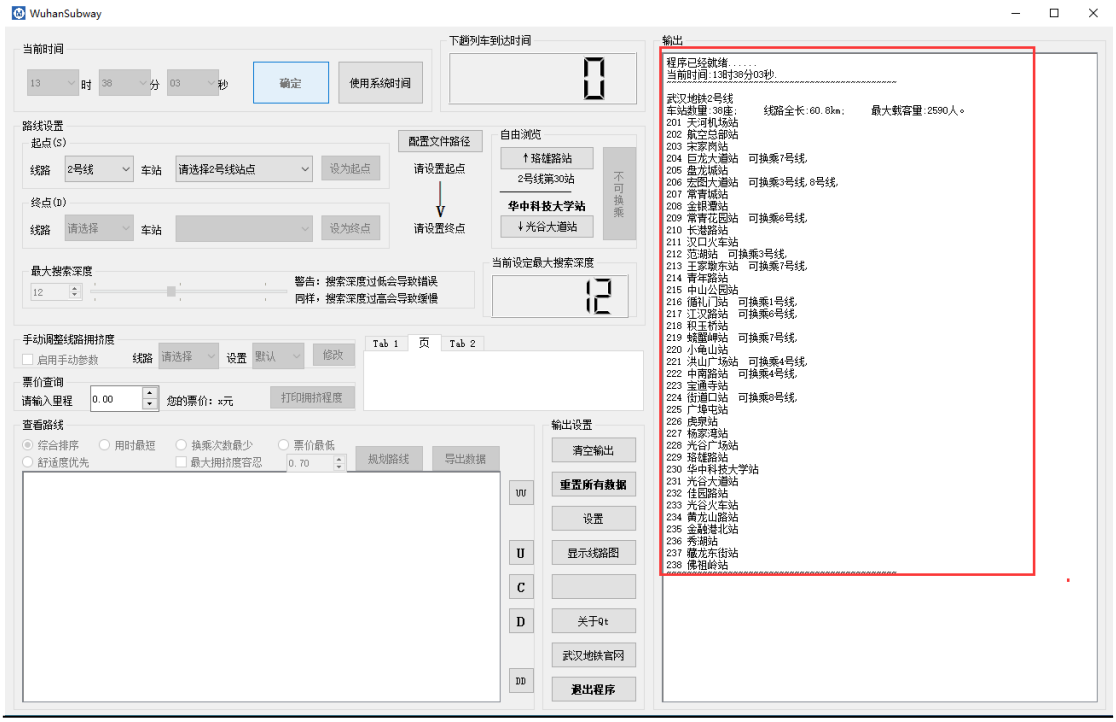

图 7-1 输出线路信息

7.2.2 显示站点信息

继续在刚刚的基础上选择 2 号线的华中科技大学站，可以看到右侧多出了站点的信息，此外，上方下趟列车到达时间显示下趟列车还有多久会到达，如图 7-2 红框所示。

图 7-2 显示站点信息

7.2.3 根据里程查询票价

在程序主界面中间靠下的位置选择票价查询功能，在数据调整框中选择需要的里程，在其右侧会显示出对应的票价，如图 7-3 红色框所示。

图 7-3 根据里程查询票价

7.2.4 地铁自由浏览

在程序主界面中间靠上的位置选择自由浏览功能，可以从设定的起点站开始任意游览地铁线路，如图 7-4 所示。

图 7-4 自由浏览线路

由于华中科技大学站不是换乘车站，因此右侧显示不可换乘按钮，且不可按动。这时我们沿 2 号线向天河机场方向走到街道口站，可以发现程序显示可换乘 8 号线，如图 7-5 所示。

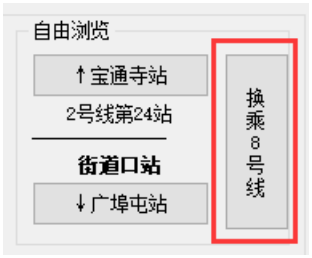

图 7-5 换乘站显示

点击换乘 8 号线即可换乘到 8 号线的站台上，并在程序主界面右侧展示地铁 8 号线的信息，如图 7-6 所示。

图 7-6 街道口站换乘 8 号线

走到 8 号线的终点站军运村站，显示如图 7-7 所示。

图 7-7 终点站显示

7.2.5 时刻表查询

在最上方及主界面右侧显示时刻表信息，如图 7-8 所示。

图 7-8 时刻表信息显示

7.2.6 继续选定终点

选择终点站为 4 号线的凤凰路站，如图 7-9 所示。

图 7-9 设置终点

关闭手动改拥挤度选项，默认最大搜索深度为 12，进入下一步。

7.2.7 以综合排序模式推荐路线

选择查看路线的综合排序模式，点击规划路线按钮，路线输出在程序主界面下方的输出框内，如图 7-10 所示。

图 7-10 按综合排序推荐路线

推荐的路线如下：

```
153 ======================路线推荐=======================
154 出发时间: 13:38
155 华中科技大学站 -------> 凤凰路站
156 以综合排序最优方式推荐路线
157
158 推荐路线 1:
159 用时: 01 时 27 分 30 秒;
160 票价: 8 元;
161 换乘次数: 1 次;
162 总里程: 39.827 公里;
163 宽松区间: 0 条, 一般区间: 28 条, 拥挤区间: 0 条.
164 总体拥挤指数: 0.200
165 预计到达时间: 15:05
166 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->中南路站--2 号线)-{换乘 4 号线}-(4 号线---->中南路站-->梅苑小区站-->武昌火车站站-->首义路站-->复兴路站-->拦江路站-->钟家村站-->汉阳火车站站-->五里墩站-->七里庙站-->十里铺站-->王家湾站-->玉龙路站-->永安堂站-->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号线)-终点
167
168 推荐路线 2:
169 用时: 01 时 33 分 30 秒;
170 票价: 9 元;
171 换乘次数: 1 次;
172 总里程: 42.7784 公里;
173 宽松区间: 0 条, 一般区间: 30 条, 拥挤区间: 0 条.
174 总体拥挤指数: 0.200
175 预计到达时间: 15:11
176 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->洪山广场站--2 号线)-{换乘 4 号线}-(4 号线---->洪山广场站-->中南路站-->梅苑小区站-->武昌火车站站-->首义路站-->复兴路站-->拦江路站-->钟家村站-->汉阳火车站站-->五里墩站-->七里庙站-->十里铺站-->王家湾站-->玉龙路站-->永安堂站-->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号线)-终点
177
178 推荐路线 3:
179 用时: 01 时 37 分 00 秒;
```

```
180 票价: 9 元;
181 换乘次数: 2 次;
182 总里程: 46.4377 公里;
183 宽松区间: 5 条, 一般区间: 27 条, 拥挤区间: 0 条.
184 总体拥挤指数: 0.169
185 预计到达时间: 15:15
186 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->江汉路
    站--2 号线)-{换乘 6 号线}-(6 号线---->江汉路站-->六渡桥站-->汉正街站-->武胜
    路站-->琴台站-->钟家村站--6 号线)-{换乘 4 号线}-(4 号线---->钟家村站-->汉阳
    火车站站-->五里墩站-->七里庙站-->十里铺站-->王家湾站-->玉龙路站-->永安堂站-
    ->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号
    线)-终点
187
```

我们记住综合推荐模式下程序推荐的第一条路线，是坐 2 号线到中南路站倒 4 号线直达凤凰路站。为了验证程序的功能及可靠性，我们选择百度地图作为验证对照组，百度地图推荐的路径如图 7-11 所示。

图 7-11 百度地图推荐的华中科技大学站~凤凰路站路线

我们发现，程序推荐的路线和百度地图推荐的路线完全一致，说明了程序的可靠性强。

此外，程序还按照综合的排序分析方法推荐了另外两条路线，我们发现这两条路线都是距离更长、耗时更久的路线，也说明了程序能够正确推荐并排序路线。

7.2.8 以用时最短模式推荐路线

选择用时最短模式，输出如图 7-12 所示。

图 7-12 以用时最短模式推荐路线

7.2.9 以换乘次数最少模式推荐路线

选择换乘次数最少模式，输出如图 7-13 所示。

图 7-13 以换乘次数最少模式推荐路线

7.2.10 以票价最低模式推荐路线

选择票价最低模式，输出如图 7-14 所示。

图 7-14 以票价最低模式推荐路线

7.2.11 以舒适度优先模式推荐路线

选择舒适度优先模式，输出如图 7-15 所示。

图 7-15 以舒适度优先模式推荐路线

我们注意到以舒适度优先模式推荐路线时，推荐的第一条路线与之前的几个模式推荐的第一条路线有一定的区别，这并不是算法出了问题，而是这个模式下仅考虑路线综合拥挤度这一个因素（对应现实中孕妇、老弱病残等人群难以承受高度拥挤的人群需要选择不那么拥挤的路线，但对于时间或绕路等需求不明显），因此会推荐这样一条路线。

7.2.12 路线推荐时增加拥挤度容忍限定

在 7.2.7~7.2.11 节中的任意模式下，均可以结合最大容忍程度设置筛选路线。用户可以任意指定 0~1.0 中任意值作为最大容忍拥挤度，推荐的路线均不会超过此设定值，并且还会按照先前的推荐规则进行排序输出。为了实现对照，我们还是以综合排序模式为基准，原先综合排序推荐的路线如下：

```
======================路线推荐======================
出发时间:  13:38
华中科技大学站 -------> 凤凰路站
以综合排序最优方式推荐路线

推荐路线 1:
用时: 01 时 27 分 30 秒;
票价: 8 元;
换乘次数: 1 次;
总里程: 39.827 公里;
宽松区间: 0 条, 一般区间: 28 条, 拥挤区间: 0 条.
总体拥挤指数: 0.200
预计到达时间: 15:05
起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->中南路站--2 号线)-{换乘 4 号线}-(4 号线---->中南路站-->梅苑小区站-->武昌火车站站-->首义路站-->复兴路站-->拦江路站-->钟家村站-->汉阳火车站站-->五里墩站-->七里庙站-->十里铺站-->王家湾站-->玉龙路站-->永安堂站-->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号线)-终点

推荐路线 2:
用时: 01 时 33 分 30 秒;
票价: 9 元;
换乘次数: 1 次;
总里程: 42.7784 公里;
宽松区间: 0 条, 一般区间: 30 条, 拥挤区间: 0 条.
总体拥挤指数: 0.200
预计到达时间: 15:11
起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->洪山广场站--2 号线)-{换乘 4 号线}-(4 号线---->洪山广场站-->中南路站-->梅苑小区站-->武昌火车站站-->首义路站-->复兴路站-->拦江路站-->钟家村站-->汉阳火车站站-->五里墩站-->七里庙站-->十里铺站-->王家湾站-->玉龙路站-->永安堂站-->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号线)-终点

推荐路线 3:
用时: 01 时 37 分 00 秒;
票价: 9 元;
换乘次数: 2 次;
总里程: 46.4377 公里;
宽松区间: 5 条, 一般区间: 27 条, 拥挤区间: 0 条.
总体拥挤指数: 0.169
预计到达时间: 15:15
起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->江汉路站--2 号线)-{换乘 6 号线}-(6 号线---->江汉路站-->六渡桥站-->汉正街站-->武胜路站-->琴台站-->钟家村站--6 号线)-{换乘 4 号线}-(4 号线---->钟家村站-->汉阳火车站站-->五里墩站-->七里庙站-->十里铺站-->王家湾站-->玉龙路站-->永安堂站-->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号线)-终点

```

我们选定 0.18 为最大容忍拥挤程度，按照如图 7-16 红框所示进行操作和选择，我们得到了如下结果。

图 7-16 添加拥挤度最大容忍程度

推荐的路线如下：

```
223 ======================路线推荐======================
224 出发时间:  13:38
225 华中科技大学站 -------> 凤凰路站
226 以综合排序最优方式推荐路线
227
228 推荐路线 1:
229 用时: 01 时 37 分 00 秒;
230 票价: 9 元;
231 换乘次数: 2 次;
232 总里程: 46.4377 公里;
233 宽松区间: 5 条, 一般区间: 27 条, 拥挤区间: 0 条.
234 总体拥挤指数: 0.169
235 预计到达时间: 15:15
236 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->江汉路站--2 号线)-{换乘 6 号线}-(6 号线---->江汉路站-->六渡桥站-->汉正街站-->武胜路站-->琴台站-->钟家村站--6 号线)-{换乘 4 号线}-(4 号线---->钟家村站-->汉阳火车站站-->五里墩站-->七里庙站-->十里铺站-->王家湾站-->玉龙路站-->永安堂站-->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号线)-终点
237
238 推荐路线 2:
239 用时: 01 时 44 分 00 秒;
240 票价: 9 元;
241 换乘次数: 4 次;
242 总里程: 46.189 公里;
243 宽松区间: 5 条, 一般区间: 27 条, 拥挤区间: 0 条.
244 总体拥挤指数: 0.169
245 预计到达时间: 15:22
246 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->中南路站--2 号线)-{换乘 4 号线}-(4 号线---->洪山广场站--4 号线)-{换乘 2 号线}-(2 号线---->江汉路站--2 号线)-{换乘 6 号线}-(6 号线---->江汉路站-->六渡桥站-->汉正街站-->武胜路站-->琴台站-->钟家村站--6 号线)-{换乘 4 号线}-(4 号线---->钟家村站-->汉阳火车站站-->五里墩站-->七里庙站-->十里铺站-->王家湾站-->玉龙路站-->永安堂站-->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号线)-终点
```

```
247
248 推荐路线 3:
249 用时: 02 时 00 分 12 秒;
250 票价: 10 元;
251 换乘次数: 4 次;
252 总里程: 53.914 公里;
253 宽松区间: 9 条, 一般区间: 30 条, 拥挤区间: 0 条.
254 总体拥挤指数: 0.154
255 预计到达时间: 15:38
256 起点-(2 号线---->华中科技大学站-->珞雄路站-->光谷广场站-->杨家湾站-->中南路站--2 号线)-{换乘 4 号线}-(4 号线---->中南路站-->梅苑小区站-->武昌火车站站-->首义路站-->复兴路站-->拦江路站-->钟家村站--4 号线)-{换乘 6 号线}-(6 号线---->钟家村站-->马鹦路站-->建港站-->前进村站-->国博中心北站-->国博中心南站-->老关村站-->江城大道站-->车城东路站-->东风公司站--6 号线)-{换乘 3 号线}-(3 号线---->东风公司站-->体育中心站-->三角湖站-->汉阳客运站-->四新大道站-->王家湾站--3 号线)-{换乘 4 号线}-(4 号线---->王家湾站-->玉龙路站-->永安堂站-->孟家铺站-->黄金口站-->新天站-->集贤站-->知音站-->新农站-->凤凰路站--4 号线)-终点
257
```

我们发现，推荐的路线整体拥挤程度确实都满足要求。

7.2.13 手动设置拥挤度

按照图 7-17 红框中所示操作可以手动设置拥挤度。

图 7-17 手动设置拥挤度

然后，打印全部线路拥挤度信息，如图 7-18 右侧红框中所示。

图 7-18 打印各线路拥挤程度

此时再点击规划路线，发现推荐的路线拥挤都发生了变化，如图 7-19 所示。

图 7-19 路线拥挤度发生变化

7.3 测试结果

综合上述测试过程，对照前述程序设计目标，得出最终的测试结果：程序能够较好地满足客户的需求，甚至还能完成用户需求之外的功能。

7.4 复杂度分析

在该程序中 DFS 路线搜索的复杂度最高，其时间复杂度可以达到：

$$T(n) = O(Cn^4)$$

其中n表示站点总数目，C为常系数。

考虑到该程序中其他模块的时间复杂度远小于此，该程序的时间复杂度可以认为就是$T(n) = O(Cn^4)$。

八、总结

8.1 整体体会

通过本次课程设计，我深刻地认识到了社会公共信息安全的重要意义。假如地铁的信息系统遭到黑客入侵，黑客可以自行修改当前地铁系统各段路线的拥挤程度，甚至能够截获用户的数据包，窥窃用户信息。为了确保信息的安全性，软件特别是公共服务的软件一定要将首要任务定为保证信息安全。很多人对软件的安全性不以为然，认为软件只要实现了用户所需的功能就可以了，然而本次课设告诉了我，软件的安全性也是衡量软件优劣程度的一个重要因素。

在实现程序的过程当中，我充分地体会到了科学家精神和工匠精神。当我在遇到程序的错误时，我发挥了科学家精神，一步一步地调试，耐心地跑完一个又一个的循环，最终成功发现了问题。我还发挥工匠精神，从用户的角度出发，完成各种各样人性化的设计，并以极高的耐心和细心来打磨最终的程序，在每个细节处都不断地进行优化。

8.2 经验与教训

在本次课程设计中，我收获了很多，也巩固了许多之前学过的知识。最大的收获时学会了使用 Qt 制作程序可视化交互界面，并了解了许多 C++面向对象编程的思想。此外，我还巩固了多文件编程、C++ STL、文件读写、图算法和数据结构等之前学过的知识，将他们运用到了实践当中，真正地化为自己的知识。

然而，本次课设也有不少的经验教训，比如我一开始在采用 DFS 搜索路径时没有合理地使用剪枝，导致搜索时间过长，且多搜出来的那些路径大多都再后面被抛弃掉，非常浪费。再比如在编写程序之前没有对各模块做很充足的规划和安排，导致在编写程序的过程中有一些函数被放在了错误的模块当中。

此外，本次课设还有一些不足之处。比如在写本次设计时，很多数据已经做了简化，但是自己的程序还是会有几个的不符合要求的地方，那么其实在真正的工程中肯定是不能做这么多简化的，所以借此机会要严格要求自己，切实进一步提高工程架构能力，其次要提升自己的调试能力，在本次程序设计过程中，由于变量，函数数目过多，在产生 bug 时，自己设置断点的调试水平很差，只能一步

步看代码去解决 bug，效率很低，还有一点是自己的测试能力不好，测试岗位也是很重要的岗位，如何选择“怪异”的测试点去测试你的程序也是技术活，自己在调试程序时采用的测试样例没有出问题，但是跑别人的测试样例时发现了问题，就说明自己在测试程序时没有精准把控程序的边界点，测试水平有待提高。

8.3 思维模式的收获和体会

我认为我在工程思维上有了突破。之前不管是写 C 还是 C++，无论是小程序还是相对较大的程序，都是一个 cpp，一个文件从头到尾结束，最多将很多抽象出来的函数分开写，从来没有写过抽象出多个模块，写多个文件的工程，在这里的工程思维是确实得到升华的。

我还在本次课设当中锻炼了自己求解问题的能力和主动的设计思维。在之前，我写代码求解的问题都很明确，题目会告诉我输入什么数据，希望能输出得到什么结果，在问题上也会很详细地描述清楚，甚至还会有思路提示或告诉我需要使用哪一类算法。但是，这次的课程设计并没有很具体很详细的要求，可能只是告诉我们一个大致的设计目标，以及希望实现的一些功能，需要我们自己分析需求，然后再利用合适的算法或方法来实现这些目标，我认为在这期间我的求解能力得到了很大的提升和锻炼。此外，还要发会主动的设计思维。这体现在一些要求中并没有涉及到的方面，但是为了精益求精和人性化设计，需要自己发挥主观能动性，主动地去设计一些甲方并没有要求的功能。

8.4 信息安全发面的考虑

我对信息安全方面的考虑是，对导出的文件进行加密，使得导出数据文件仅能在程序内置的解码器下才能显示正常的信息，防止无权限用户对数据进行窃取。

九、参考文献

[1] 曹计昌，卢萍，李开. C 语言与程序设计. 电子工业出版社，2013

[2]严蔚敏等.数据结构（C 语言版）. 清华大学出版社，

[3] Larry Nyhoff. ADTs, Data Structures, and Problem Solving with C++. Second Edition, Calvin College, 2005

[4] 殷立峰. Qt C++跨平台图形界面程序设计基础. 清华大学出版社，2014:192～197

[5] 严蔚敏等.数据结构题集（C 语言版）. 清华大学出版社

[6] 胡凡，曾磊. 算法笔记. 机械工业出版社，2018

十、主要程序片段

```
#include "define.h"
#include "mapping.h"
Line line[MAX_LINE_NUM];
bool openline[MAX_LINE_NUM];
vector<StArray> AdjLst;

//函数名:   input
//作者:     袁也
//日期:     2021/02/16
//功能:     从文件中将数据读入内存
//输入参数:  path 数据文件所在路径
//返回值:   状态值(int)
//         返回 INFEASTABLE 表示文件打开失败;
//         返回 OK 表示文件读取成功。
//修改记录:
status input(const char* path)
{
    ifstream in;
    in.open(path);
    if (in.is_open()==false)
    {
        return INFEASTABLE;
    }
    else
    {
        int line_id;
        while (!in.eof())//遍历存储全部线路
        {
            in >> line_id;
            openline[line_id] = true;
          line[line_id].cong_flag=0;
            in >> line[line_id].fullNum >> line[line_id].length >> line[line_id].St_Num;//从文件读取线路信息
            for (int i = 1; i <= line[line_id].St_Num; i++)//遍历该线路的全部车站
            {
                Station st;//创建 Station 结构体临时存放车站信息
                st.id.line = line_id;
                in >> st.id.station_number >> st.name;
                int temp;//暂存输入的可换乘线路编号
                in >> temp;
                if (temp == 0)//该车站不是换乘站，没有其他线路，直接压入 Vector
                {
                    st.transfer = false;//该站不是换乘站
                    line[line_id].st_list.push_back(st);
                }
                else//该站是换乘车站
                {
                    st.transfer = true;//该站是换乘站
                    st.TransTo.push_back(temp);
                    for (;;)//遍历该站可换乘的全部线路
                    {
                        in >> temp;
                        if (temp != 0)
                            st.TransTo.push_back(temp);
                        else
                            break;
                    }
                    line[line_id].st_list.push_back(st);
                }
            }
        }
        in.close();
        return OK;
    }
}
```

```
322
323 //函数名:   Test_Output
324 //作者:     袁也
325 //日期:     2021/02/16
326 //功能:     该函数仅用于测试!
327 //            将内存中存储的全部信息按照特定的的格式打印出来
328 //输入参数:  无
329 //返回值:   状态值(int)
330 //         返回 OK 表示打印成功。
331 //修改记录:
332 status Test_Output(void)
333 {
334     for (int i = 1; i < MAX_LINE_NUM; i++)
335     {
336         if (!openline[i])
337             continue;
338        printf("武汉地铁%d 号线:\n", i);
339        printf("线路信息:\n");
340        printf("\t 载客量:%d;\n", line[i].fullNum);
341        printf("\t 线路全长:%gkm;\n", line[i].length);
342        printf("\t 车站数目:%d;\n\n", line[i].St_Num);
343         vector<Station>::iterator itSt;
344        printf("\t 车站编号\t 名称\t\t 是否为换乘站\t 换乘线路\n");
345         for (itSt = line[i].st_list.begin(); itSt !=
    line[i].st_list.end(); itSt++)
346         {
347           printf("\t%d%02d\t", itSt->id.line,
    itSt->id.station_number);
348           cout << itSt->name;
349           printf("\t\t");
350              if (itSt->transfer)
351              {
352               printf("是\t\t");
353                   for (vector<int>::iterator itInt =
    itSt->TransTo.begin(); itInt != itSt->TransTo.end(); itInt++)
354                   {
355                   printf("%d 号线 ", *itInt);
356                   }
357                   putchar('\n');
358              }
359              else
360              {
361               printf("否\t\t-------\n");
362              }
363         }
364     }
365     return OK;
366 }
367
368 //函数名:   CreateStationNode
369 //作者:     袁也
370 //日期:     2021/02/17
371 //功能:     建立邻接表中的车站节点
372 //输入参数:  车站信息结构体
373 //返回值:   车站节点指针
374 //修改记录:
375 StNode* CreateStationNode(Station s)
376 {
377     StNode* ret = new StNode;
378     ret->next = NULL;
379     ret->station = s;
380     return ret;
381 }
382
383 //函数名:   BuildMap
384 //作者:     袁也
385 //日期:     2021/02/16
386 //功能:     通过内存中的数据建立无向图
387 //输入参数:  无
388 //返回值:   状态值(int)
389 //         返回 OK 表示转换成功。
390 //修改记录:
```

```
391 status BuildMap(void)
392 {
393     for (int i = 1; i < MAX_LINE_NUM; i++)
394     {
395         if (!openline[i])
396             continue;
397         vector<Station>::iterator itSt;
398         for (itSt = line[i].st_list.begin(); itSt !=
    line[i].st_list.end(); itSt++)
399         {
400             vector<Station>::iterator it_temp;//临时迭代器
401             int count = 0;//邻接结点计数器
402             StArray nowinArray;
403             nowinArray.station = *itSt;
404             StNode* FirstStNode = new StNode;
405             StNode* tail = FirstStNode;
406             if (itSt != line[i].st_list.begin())//向邻接表中链入上一站
    数据（如果有）
407             {
408                 it_temp = itSt - 1;
409                 count++;
410                 tail->next = CreateStationNode(*it_temp);
411                 tail = tail->next;
412             }
413             if (itSt != line[i].st_list.end() - 1)//向邻接表中链入下一
    站数据（如果有）
414             {
415                 it_temp = itSt + 1;
416                 count++;
417                 tail->next = CreateStationNode(*it_temp);
418                 tail = tail->next;
419             }
420             if (itSt->transfer == true)//向邻接表中链入换乘车站数据（如
    果有）
421             {
422                 vector<int>::iterator itInt;
423                 for (itInt = itSt->TransTo.begin(); itInt !=
    itSt->TransTo.end(); itInt++)
424                 {
425                     vector<Station>::iterator temp_itSt;
426                     for (temp_itSt = line[*itInt].st_list.begin();
    temp_itSt != line[*itInt].st_list.end(); temp_itSt++)
427                     {
428                         if (temp_itSt->name == itSt->name)
429                         {
430                             count++;
431                             tail->next =
    CreateStationNode(*temp_itSt);
432                             tail = tail->next;
433                         }
434                     }
435                 }
436             }
437             tail->next = NULL;
438             nowinArray.numOfAdjacentNodes = count;
439             nowinArray.next = FirstStNode->next;
440             AdjLst.push_back(nowinArray);
441         }
442     }
443     return OK;
444 }
445
446 vector<Station>::iterator FindStation(int lineid, int stid)
447 {
448     vector<Station>::iterator ret;
449     for (ret = line[lineid].st_list.begin(); ret !=
    line[lineid].st_list.end(); ret++)
450     {
451         if (ret->id.station_number == stid)
452             break;
453     }
454     return ret;
455 }
456
```

```
457 //函数名:   PrintAdjList
458 //作者:     袁也
459 //日期:     2021/02/18
460 //功能:     #该功能仅为测试使用! #
461 //            打印内存中的邻接表
462 //输入参数:  无
463 //返回值:   状态值(int)
464 //         返回 OK 表示邻接表打印成功。
465 //修改记录:
466 status PrintAdjList(void)
467 {
468     vector<StArray>::iterator itAdjLst;
469     for (itAdjLst = AdjLst.begin(); itAdjLst != AdjLst.end();
    itAdjLst++)
470     {
471         printf("\t%d%02d ", itAdjLst->station.id.line,
    itAdjLst->station.id.station_number);
472         cout << itAdjLst->station.name;
473          StNode* p;//邻接车站遍历指针
474          putchar('\t');
475          for (p = itAdjLst->next; p != NULL; p = p->next)
476          {
477            printf("\t%d%02d ", p->station.id.line,
    p->station.id.station_number);
478            cout << p->station.name;
479          }
480          putchar('\n');
481     }
482     return OK;
483 }
484
```