**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: 武汉地铁路径规划系统

班 级： 信息安全1901班

学 号： U201911672

姓 名： 唐 鼎

成 绩：

指导教师： 朱建新

**完成日期：** 2021**年 3月 6 日**

**目录**

[一、系统需求分析 1](#_Toc67387072)

[二、总体设计 2](#_Toc67387073)

[三、数据结构设计 3](#_Toc67387074)

[四、详细设计 5](#_Toc67387075)

[4.1 路径规划算法 5](#_Toc67387076)

[4.2 哈希表 6](#_Toc67387077)

[4.3路径规划具体实现 7](#_Toc67387078)

[五、系统实现 9](#_Toc67387079)

[六、运行测试与结果分析 13](#_Toc67387080)

[6.1界面介绍 13](#_Toc67387081)

[6.2测试 15](#_Toc67387082)

[复杂度分析 17](#_Toc67387083)

[七、总结 18](#_Toc67387084)

[八、参考文献 19](#_Toc67387085)

[九、程序使用说明 20](#_Toc67387086)

# 一、系统需求分析

本次课程设计要求我们实现一个武汉地铁路径规划系统，客户可以任意选择武汉市地铁站的两个站点作为出发站和目的站点，要求我们的程序能为用户匹配出1-3条最优的地铁乘车路线以及换乘建议。

除此之外，为了给客户良好的使用体验，以及为了使产品更加符合实际，产品需求中还有一些额外要求：

1. 用户可以选择地图上的任意一个始发站点，然后浏览其相邻站点，或在相邻站点换线，实现站点浏览。
2. 利用实验需求中的简化约定为武汉地铁每个站的不同线路设置乘车时间表，以方便用户了解最近一班车什么时候到、什么时候离开。
3. 得到的多条线路按票价排序，以方便用户找到最适合自己的线路
4. 模拟不同时间段武汉地铁各线路的拥挤程度，并帮助用户合理避开拥挤度过高的乘车方案

# 二、总体设计

经过对网上其它类似项目实现框架的参考以及本人自己的再三考虑。最终决定使用QT框架来实现本次项目，利用C++完成项目客户端界面的设计，纯C语言完成对项目的数据结构和算法的内部实现。

大致框架如下图

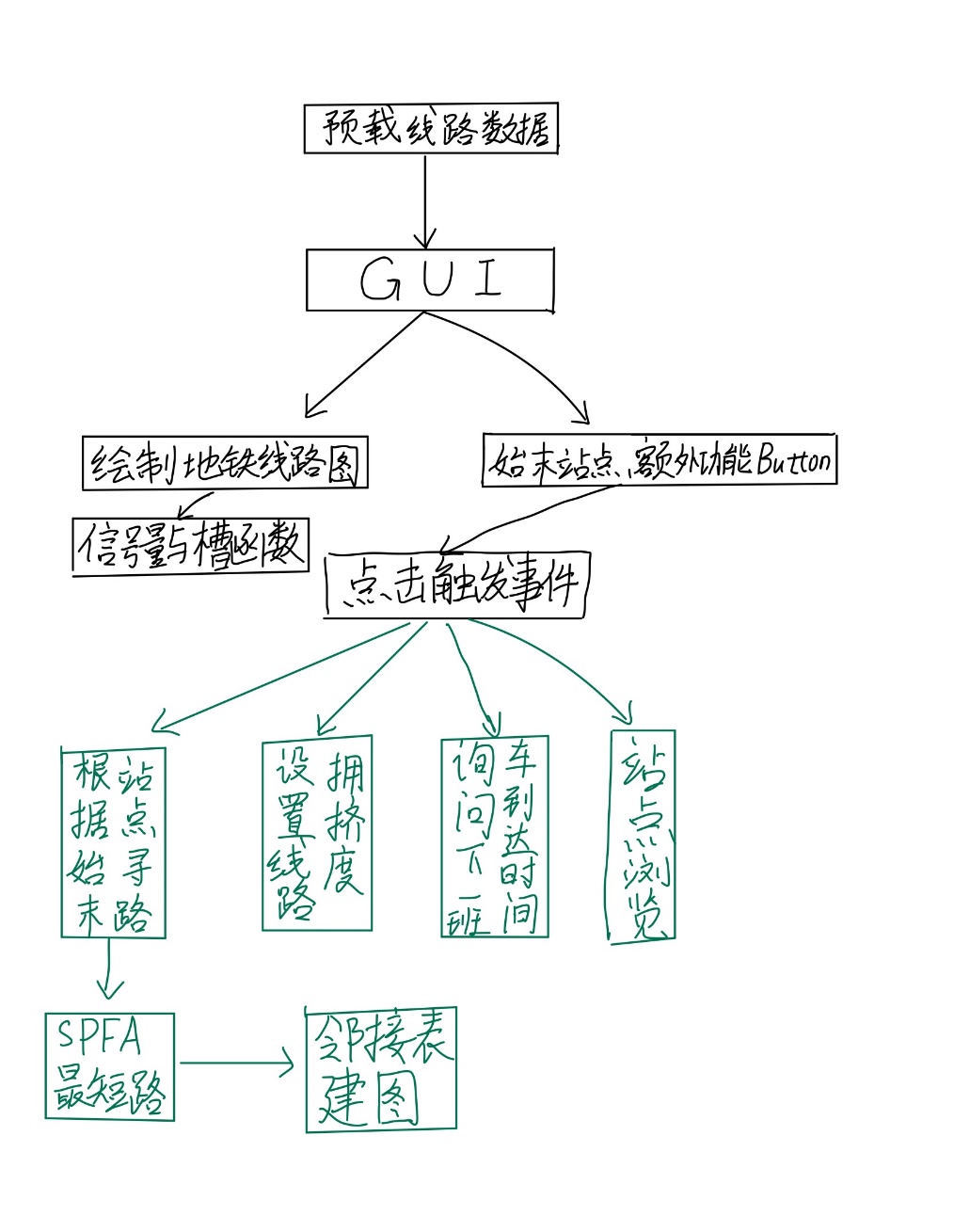


图2.1项目大致流程

**注**：标黑部分为GUI逻辑设计，标绿部分为核心算法逻辑。

# 三、数据结构设计

武汉地铁线路实际上是一个图论中的无向图的结构，除此之外，每条边属于不同的线路，一般来说一条边只属于一条地铁线，但也存在两个相邻站点有两条不同地铁线经过（如中南路和洪山广场）。经过统计、武汉地铁站一共200左右个站点，大小完全可以采用邻接矩阵存储，但为了贴合一般工程，不考虑采用。故本项目采用**链式前向星（邻接链表）**存储图的结构

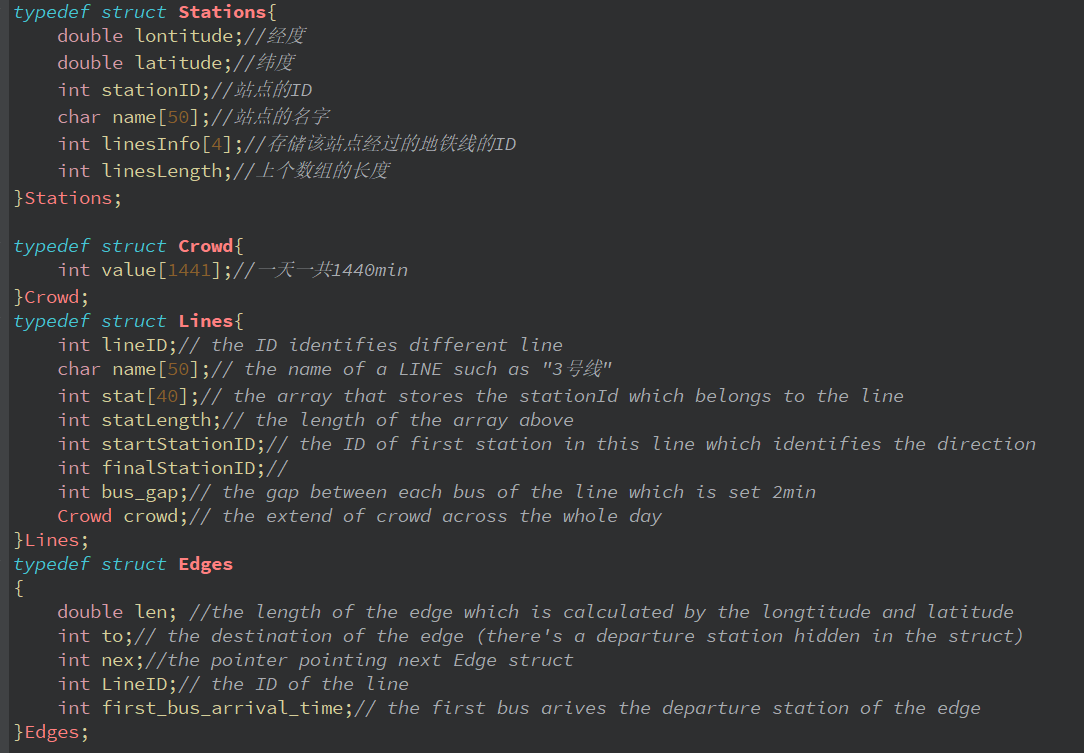


图3.1 站点、线路、边的结构体

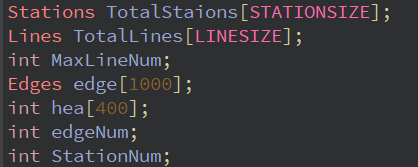


图3.2几个重要的全局变量

**解释**：三个结构体类型的变量存储的信息与其含义一致。MaxLineNum用来存储武汉地铁线路中最大的线路ID，因为武汉地铁中部分线路没有开通，我打算保持线路ID与其数组的Index一致。EdgeNum和StationNum保存其结构体数组的大小。hea[]数组是链式前向星的一部分，用来保存链表的头部，是遍历某站点所连边的入口。

Stations结构体存储了一个站点的必要信息，包括站点的经纬度, id , 经过的地铁线路和站点的名字；Lines结构体存储了一条线路的必要信息，包括线路名字, id , 所包含的所有站点的 id , 线路两个端点的站点 id ，拥挤度，每班车到达下一站点所需时间（简化约定）。Edges结构体存储了地铁线路中单向边的必要信息，每个站点所连接的边的结构体通过一个单向链表存储，每个Edges结构体都是链表的一个节点，每个站点的单向链表的表头（的 Index ）由hea数组保存，每个节点都保存了这个边的长度（由经纬度计算得到），所属的线路的 id ，目的站点 id ,第一班车到达时间（用于时刻表），以及下一个节点的 Index等。

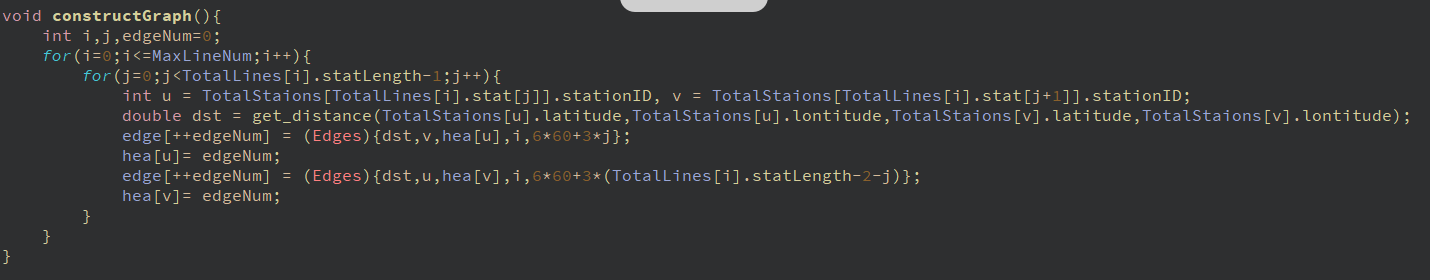


图3.3存储图结构的算法

**解释：**在读入文件并完成站点线路信息的初始化后进行构图，我采取依次遍历线路将各个线路的站点依次连接起来，可以发现，每条线路往同一方向的边的Index是同奇偶的（即同一方向的边的结构体的 Index 要么都是奇数，要么都是偶数），可依据这个特性判断一条线路上的不同方向。

# 四、详细设计

## 4.1 路径规划算法

路径规划算法采用了较为容易实现的SPFA算法，算法描述如下：

1. 首先将出发节点放入队列，打上标记，防止重复入队
2. 进入循环，结束条件为队列为空
3. 取出队列中的队首元素，去除标记，遍历并松弛它所连接的边，将松弛成功的边的目标节点放入队列
4. 重复如上循环，直到队列为空

伪代码如下：

1 q **=** **new** queue**();**

2 q**.**push**(**S**);**

3 in\_queue**[**S**] =** **true;**

4 **while** **(!**q**.**empty**()) {**

5 u **=** q**.**pop**();**

6 in\_queue**[**u**] =** **false;**

7 **for** each edge**(**u**,** v**) {**

8 **if** **(**relax**(**u**,** v**) && !**in\_queue**[**v**]) {**

9 q**.**push**(**v**);**

10 in\_queue**[**v**] =** **true;**

11 **}**

12 **}**

13 **}**

运算完以上代码，可以得到从始发站点到其它各个站点的最短路径的代价。

## 4.2 哈希表

由于输入的站点数据是按每条线路中的站点数据依次遍历的，因此输入数据可能存在重复站点，所以需要使用哈希表来实现站点名（汉字字符串）到数值的映射，用来进行数据去重，实现如下图：

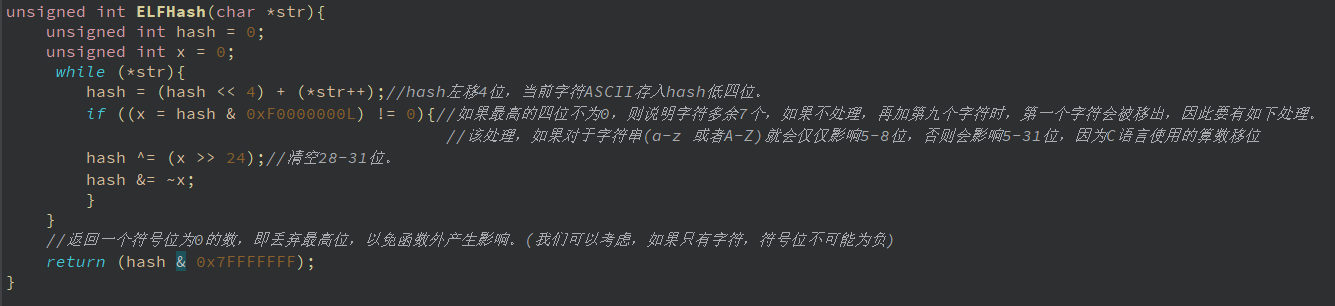


图4.1哈希函数的实现

采用了较流行的ELF哈希算法

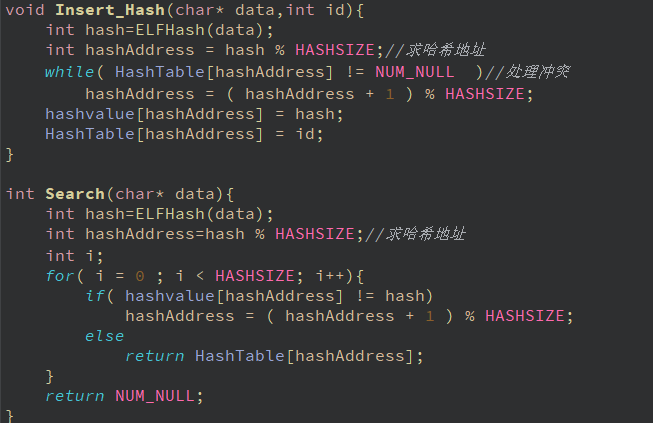


图4.2哈希表的实现

我们不能保证一个哈希函数不存在哈希碰撞（Collision resistance），因此必须要有一个方法来处理哈希碰撞。我的实现里采用了**开放地址法**，如果某个哈希值在哈希表里的位置已经被占用了，就找它的相邻位置，直到有空闲位置。由于站点id是唯一的，我使用了HashTable数组来存储站点哈希值对应的站点的id，用来辅助确认这个哈希值的唯一性。

## 4.3路径规划具体实现

项目中影响路径规划的因素比较多，对这些因素的处理都综合到SPFA里了。具体代码如下



图4.3最短路的具体实现

可以看到在算法主体中加入对其它综合因素的处理后，变得臃肿许多，图中红框框起来的部分都是对各种影响最短路因素的处理，包括拥挤度，换乘线路，以及换乘或出发时线路没有地铁停靠等。另外松弛的if分支中crowd数组用来累加总的拥挤度的和，在全部的边松弛结束后，目的站点的crowd值除以最短路路径经过的边的个数可以得到这个最短路方案的平均拥挤度；path数组用来存储最短路的路径，每次松弛成功都要更新所松弛的边的目的站点的path值，即path[v]=u这样在全部松弛结束后可以根据path[a.end]回溯的到整个最短路方案的线路;transfer数组用来存储路径中的线路id方便在松弛过程中对换线的松弛进行额外处理，time与dis数组则是到达该站点的最短时间/路径。经过一次最短路函数（SPFA算法）的处理，能保证得到一条最优解。

# 五、系统实现

开发环境配置入下：

编辑器：QT creator 4.13.1（Community）

集成开发环境：QT 5.12.10

C语言编译器：Mingw730\_64

主要函数原型：

1 // @description 根据简化约定设置线路各个时间点的拥挤度

2 // @params None

3 // @return None

4 **void** crowd\_simulation\_init**();**

5

6 // @description 读入文件中的站点数据并进行全局初始化

7 // @params None

8 // @return Mone

9 **int** readFile\_txt**();**

10

11 // @description 转换角度为弧度

12 // @params double 角度

13 // @return double 弧度

14 **double** radian**(double** d**);**

15

16 // @description 根据两个地点的敬畏度计算距离（单位为km）

17 // @params double \* 4 纬度 经度 纬度 经度

18 // @return double 距离

19 **double** get\_distance**(double** lat1**,** **double** lng1**,** **double** lat2**,** **double** lng2**);**

20

21 // @description 根据存储的线路和站点信息构建图的邻接表

22 // @params None

23 // @return None

24 **void** constructGraph**();**

25

26 // @description 转换不同线路不同时间的拥挤读为拥挤系数

27 // @params id int 线路id， time int 当前时间

28 // @return double 拥挤系数

29 **double** crowd\_index\_convert**(int** id**,int** time**);**

30

31 // @description 转换乘车距离为对应的票价

32 // @params distance int 乘车距离

33 // @return double 对应的价格

34 **double** get\_price**(int** dis**);**

35

36 // @description 获取站点在对应线路的数组索引

37 // @params u int 站点id， line int 线路id

38 // @return i int 对应的索引

39 **int** get\_index\_from\_line**(int** u**,int** line**);**

40

41 // @description 为三种策略的最短路算法进行初始化

42 // @params None

43 // @return None

44 **void** dij\_init**();**

45

46 // @description 以拥挤度加权时间最优进行spfa算法

47 // @params struct Request 完成一次路径规划服务的必要参数

48 // @return strcut Solution 路径规划的结果

49 Solution time\_min\_spfa**(**Request a**);**

50

51 // @description 以总时间最优进行spfa算法

52 // @params struct Request 完成一次路径规划服务的必要参数

53 // @return strcut Solution 路径规划的结果

54 Solution sum\_time\_min\_spfa**(**Request a **);**

55

56 // @description 以换乘次数最少为指标进行spfa算法

57 // @params struct Request 完成一次路径规划服务的必要参数

58 // @return strcut Solution 路径规划的结果

59 Solution transfer\_min\_spfa**(**Request a**);**

60

61 // @description 以票价最少（路途最短）为指标进行spfa算法

62 // @params struct Request 完成一次路径规划服务的必要参数

63 // @return strcut Solution 路径规划的结果

64 Solution price\_min\_spfa**(**Request a**);**

65

66 // @description 初始化哈希表

67 // @params None

68 // @return None

69 **void** InitHashTable**();**

70

71 // @description ELF哈希函数

72 // @params char\* 待hash的字符串

73 // @return int 得到的哈希值

74 **unsigned int** ELFHash**(char** **\***str**);**

75

76 // @description 向哈希表插入一个元素

77 // @params char\* 待插入字符串 ，id 站点id（哈希表存的值）

78 // @return None

79 //哈希表的插入函数，可用于构造哈希表

80 **void** Insert\_Hash**(char\*** data**,int** id**);**

81

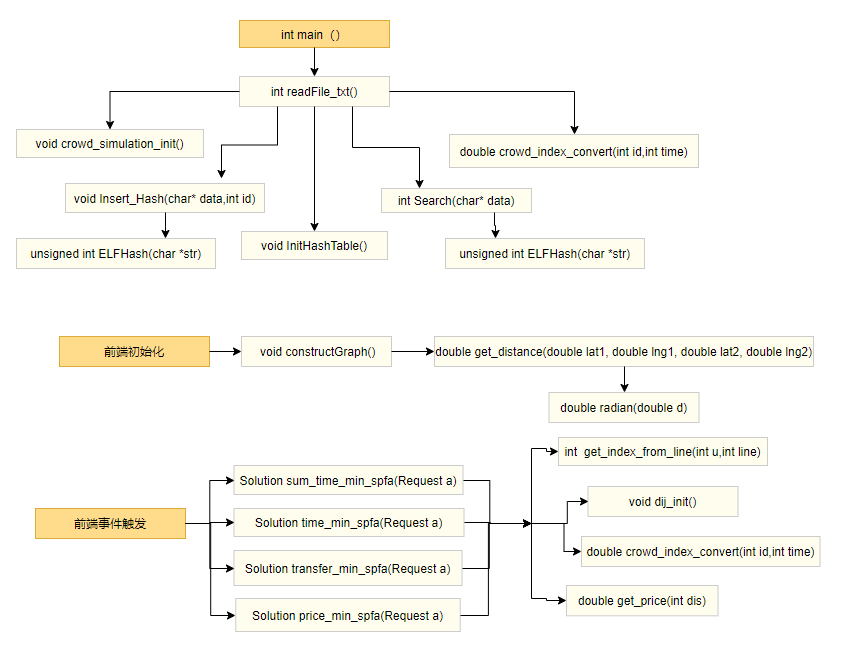
82 // @description 在哈希表中查找一个一个字符串对应的值

83 // @params char\* 待查找的字符串

84 // @return id 站点id or NUM\_NULL(未找到)

85 **int** Search**(char\*** data**);**

函数调用关系如下



# 六、运行测试与结果分析

## 6.1界面介绍

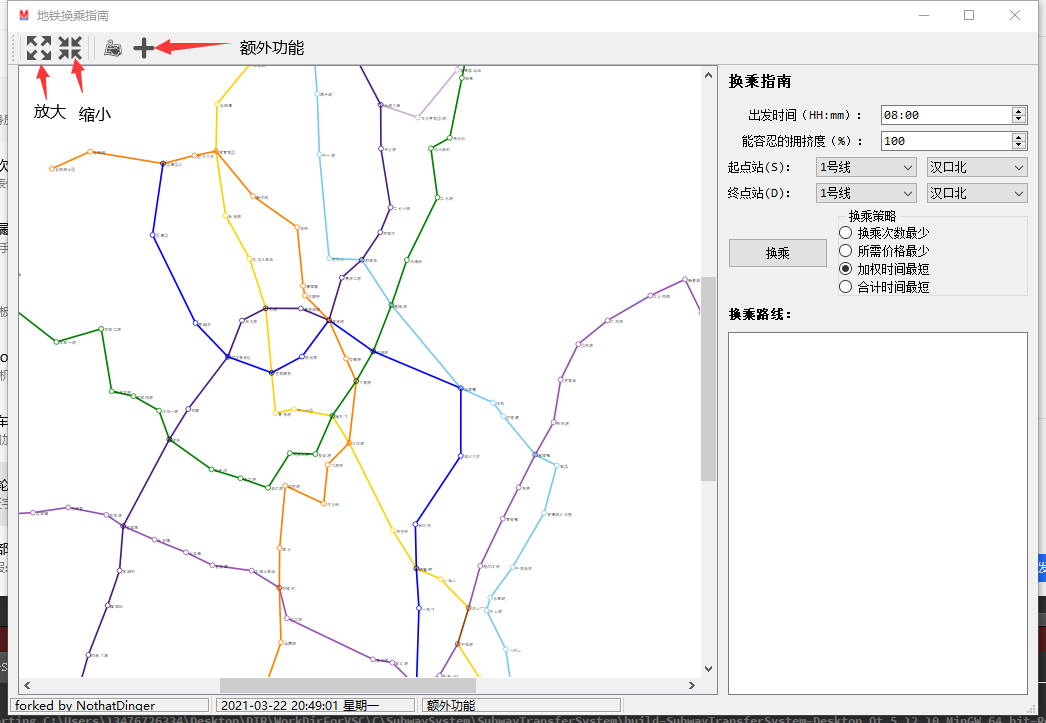


图6.1程序主题界面

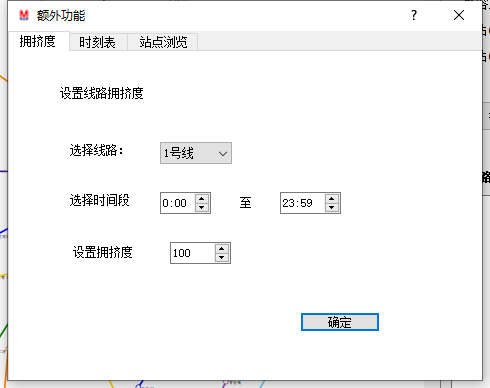
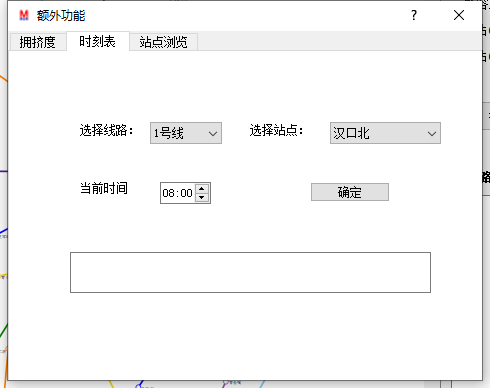
如图，右侧为路线推荐系统的功能栏，用户可以自主选择出发站点、目的站点，出发时间，拥挤状况的容忍度以及四种换乘策略。左上角有按钮以供地铁线路图的放大缩小，同时也能配合ctrl键+鼠标滚轮实现线路图的放大缩小。

图6.2额外功能界面

额外功能一栏可以为帮助模拟真实地铁线路情况，用户可以自定义设置不同线路不同时间段的拥挤度，可以知道每个站点各个方向下一班车到达时间，也可以步进浏览武汉地铁站点。

## 6.2测试

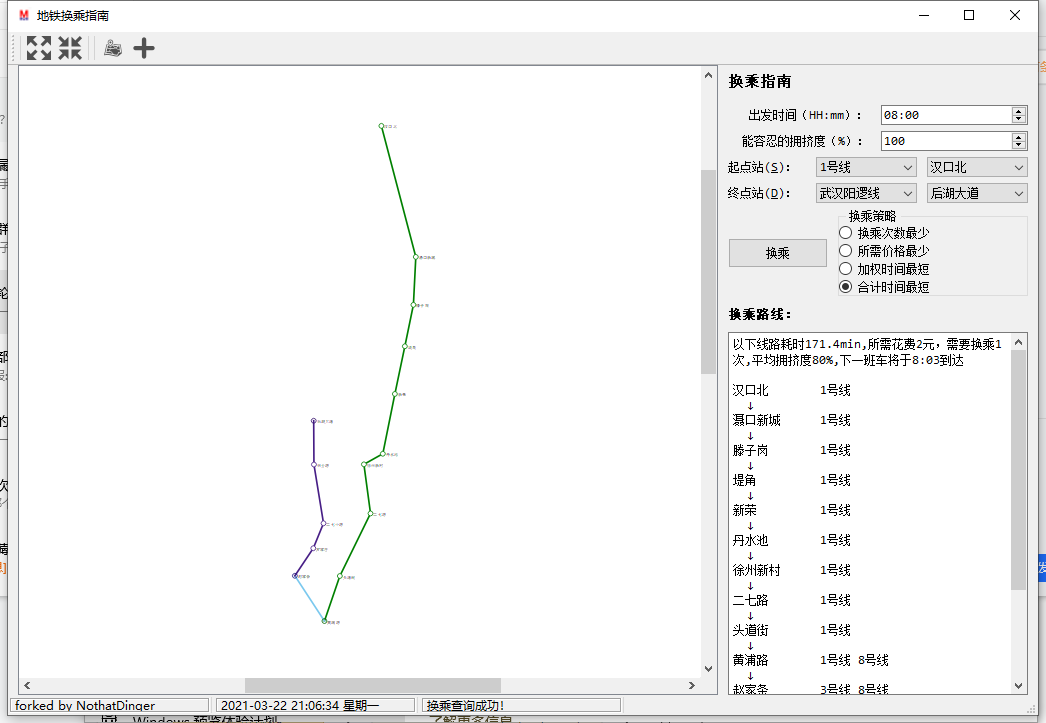


图6.3测试一

出发时间选择8：00，容忍度选择100%，选择总时间最短策略，可以正常跑出最优路径。

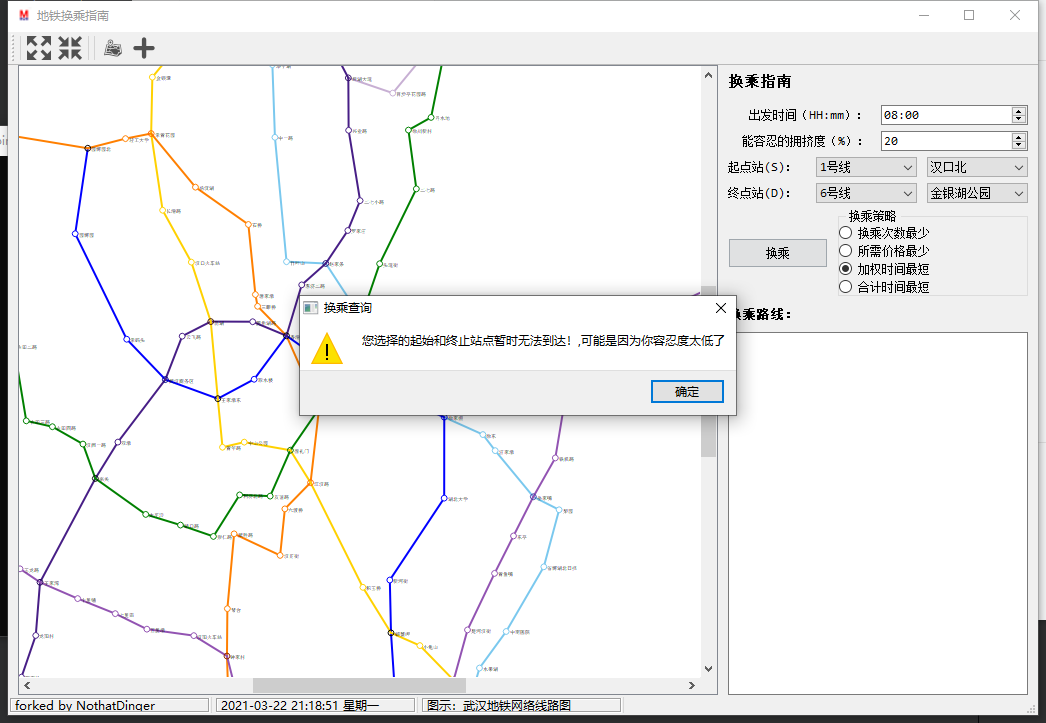


图6.4测试二

当采用较低容忍度以至于没有线路符合要求时，发生弹窗提醒。

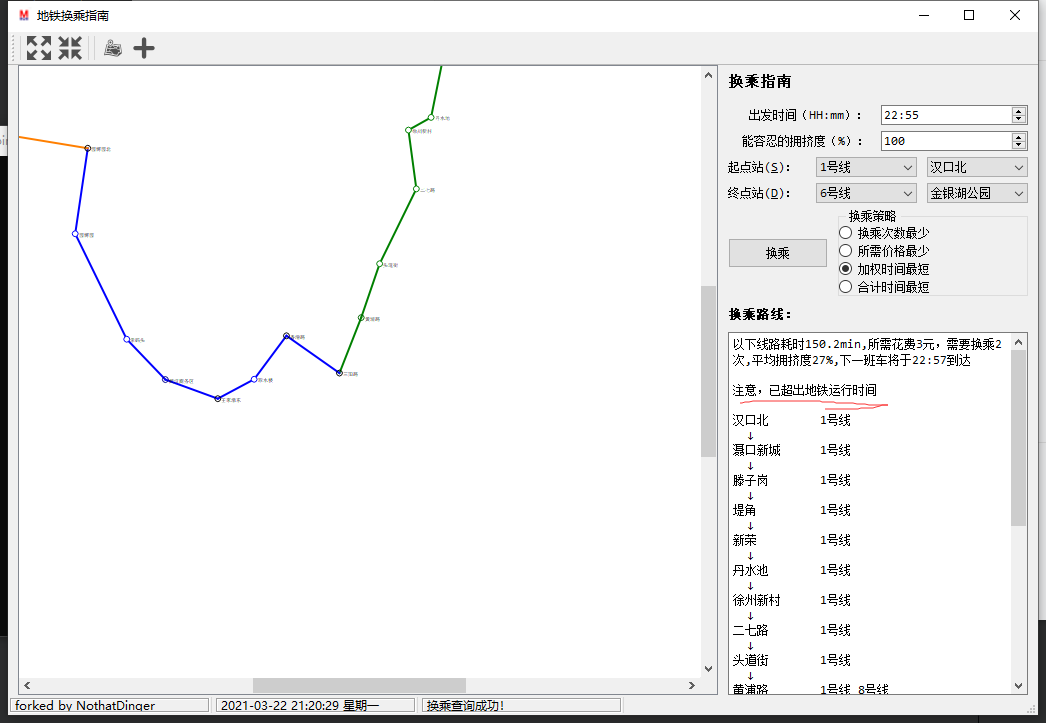


图6.5测试三

当出发时间较晚，程序仍然会跑出最优线路，但会附加提醒。

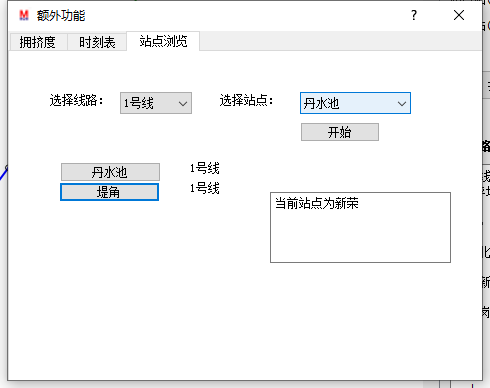
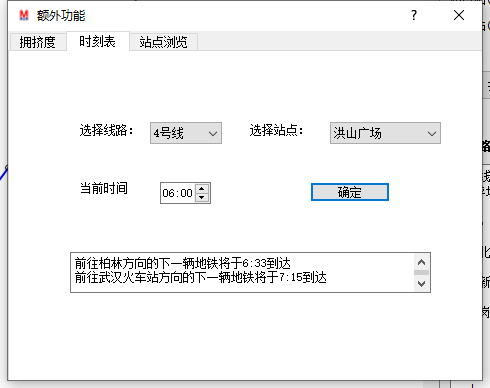
 

图6.6额外功能展示

站点浏览时，每访问一个站点，左边会显示与它相连的站点按钮，点击即进入了所点击站点，右图乘车时刻表能告诉你距离当前事件最近的该线路该站点的两个方向的车到达时间。拥挤度设置功能没有用户交互故不做展示。

## 复杂度分析

理论上SPFA算法的最坏时间复杂度是O(NM)，N为图中节点个数，M为图中边的个数，它的策略是将每个成功松弛过的点不重复的放入队列，但事实上，根据图的结构的不同，每个节点入队的次数是无法确定的，只知道在连通图中，它的入队次数大于等于1小于M,所以SPFA算法的时间复杂度是在O(N)~O(NM)不等，不够稳定。

# 七、总结

我感觉虽然明确说明了课程设计对GUI不做强制要求，但大家还是主要花费了大量的时间和精力去设计GUI。就我个人而言，我是从寒假最后几天开始写的，其实整个项目的核心是最短路/k短路算法。姑且认为考察的是最短路算法，最短路的几个算法的核心也就不到50行C++代码，因为要求核心代码只能用C语言写，砍掉了用C++的STL库优化的部分，自己手写个用于优化的数据结构或者用其他不需要STL库的算法也不需要花上一天时间。如果是命令行GUI（无法直观展示线路图），我其实第二节课就可以给老师检查了。后来检查报告单说明了图形化界面只占4分，但我观察到周围有不少同学在设计自己的图形界面，我整个人都焦虑了起来，也开始自己安装环境，从网上借鉴了一个PC客户端的图形化界面的代码， 并根据自己的代码的接口魔改借鉴的代码。

当然，为了这4分折腾了不少，有点“卷怪”的味道，但在这几天设计客户端代码的过程中我还是有了不少客户端编程的经验，我熟悉了QT框架的大致使用方法，能自主设计QT中的UI界面，并为其前端控件设计信号以及对应的槽函数等等，也可以说是有不少收获。

过程中也反映出来一些问题，比如，完成项目后，测试不够精细，导致在检查以及书写报告的过程中查出不少小bug，这也许是到了项目后期心浮气躁导致的，也提醒了我日后在编码过程中对于心态的管理。

# 八、参考文献

[1] https://zhuanlan.zhihu.com/p/130501729

[2] <https://blog.csdn.net/weixin_38293850/article/details/78352445>

[3] <https://github.com/BaiJiazm/SubwayTransferSystem/>

[4] http://c.biancheng.net/qt/

# 九、程序使用说明

为了方便老师检查，压缩文件的”build-SubwayTransferSystem-Desktop\_Qt\_5\_12\_10\_MinGW\_64\_bit-Release”的目录中打包了一个release版本的文件夹，里面的exe程序在文件夹内是可以直接运行的，因为其依赖的五个库文件被打包在文件夹内，如若需要移动必须带上这五个库文件，Qt5Core.dll, Qt5Gui.dll, Qt5Widgets.dll, libgcc\_s\_seh-1.dll, libstdc++-6.dll。

如果老师需要查看代码，并且安装了QT Creator。在QT中打开”SubwayTransferSystem.pro”即可

程序功能很简单，在前面的**运行测试**部分已经做出介绍。