一、任务描述

城市之间有三种交通工具（汽车、火车和飞机）相连，某旅客于某一时刻向系统提出旅行要求，系统根据该旅客的要求为其设计一条旅行线路并输出；系统能查询当前时刻旅客所处的地点和状态（停留城市/所在交通工具）。

二、功能需求

* 城市总数不少于10个 **[本组选取12个城市]**
* 建立汽车、火车和飞机的时刻表（航班表）
  + 有沿途到站及票价信息
  + 不能太简单（不能总只是1班车次相连）**[本组共863班车次]**
* 旅客的要求包括：起点、终点、途经某些城市和旅行策略
* 旅行策略有：
  + 最少费用策略：无时间限制，费用最少即可
  + 最少时间策略：无费用限制，时间最少即可
  + 限时最少费用策略：在规定的时间内所需费用最省
* 旅行模拟查询系统以时间为轴向前推移，每10秒左右向前推进1个小时(查询状态的请求不计时) **[本组将推进时间改为每1秒向前推进1小时]**
* 不考虑城市内换乘交通工具所需时间
* 系统时间精确到小时
* 建立日志文件，对旅客状态变化和键入等信息进行记录
* 选做一：某旅客在旅行途中可更改旅行计划，系统应做相应操作 **[本组*未完成* 换乘]**
* 选做二：用图形绘制地图，并在地图上反映出旅客的旅行过程 **[本组*已完成* 图形化]**

三、总体方案设计

1、软件开发环境

Windows10 PC + Visual Studio 2017 + Qt 5.11.3 msvc2017\_64

2、模块划分

* 主模块

接收键盘和鼠标命令，分析命令后调用相应模块，以时间轴向前推进，完成图形化信息反馈

* 需求处理模块

1. 接收主模块发送的路线规划需求，调用相应算法，给出旅游路线
2. 接收主模块发送的查询状态需求，调用相应算法，给出当前状态

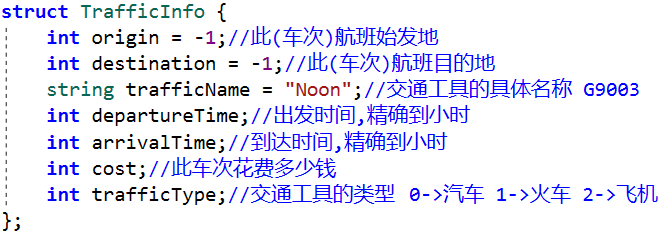
* 日志文件处理模块

提供相应接口,创建日志文件，接收要写入的字符串并写入文件

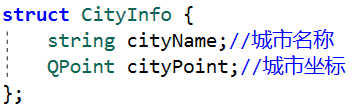
四、数据结构和重要变量说明

1、数据结构

① 车次(航班)信息



② 城市信息



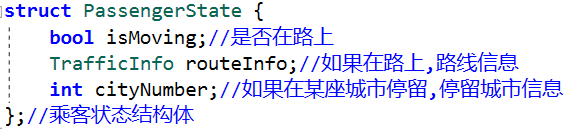
③ 城市信息列表(全局变量)



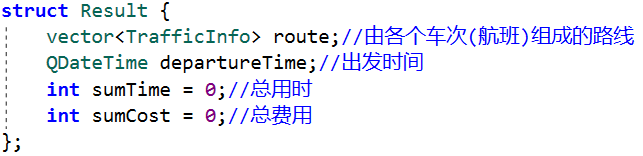
④ 车次(航班)列表(全局变量)



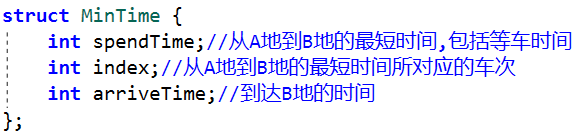
⑤ 旅客状态



⑥ 旅行路线结果



⑦ 两地之间最短时间车次(航班)信息



⑧ 邻接矩阵



2、重要变量

① 路线结果(全局变量)



储存最优路线结果，作为全局变量可以为主模块图形化和日志模块服务

② 文件I/O(全局变量)



用于创建日志，更新日志文件，作为全局变量可以在主窗口关闭时安全关闭文件

四、模块设计说明及其调用关系

1、模块设计说明

① 主模块

Ⅰ图形化

·地图的实现：以map.png为背景图

·路线的绘制：根据result和cityInfoList获取各个城市的坐标，坐标之间连线

·动画的实现：采用Qt的Animation，将动画属性设为“位置”，由城市坐标确定**动画的位置区间**，由车次(航班)时间和等待时间确定**动画的时间区间**，根据动画状态的变化(run、stop)**确定显示哪种交通方式**

Ⅱ信息获取

·利用QComoboBox确定选择的策略、起始地和目的地，QLineEdit获取限制时间

·通过判断用户是否选择了第三种策略，来确定用户输入“限制时间”的权限

Ⅲ时间轴控制

·利用QDateTime获取当前时间，利用QTimer定时器，设置超时为10ms，每超时一次QDateTime增加36s，达到1s推进1h的效果

·当程序在计算最优路径时，QTimer停止计时，计算完最优路径后，QTimer继续计时，当用户在查询状态时，由于计算时间很快，不予停止计时。

Ⅳ输入合法性

·根据用户选择的起始地和目的地，判断是否两者为同一地点，若同一地点则报错

·根据用户选择的必经城市，判断其是否和起始地、目的地重复，若重复则报错

·根据用户选择的必经城市，判断其数量，大于3个则报错(具体原因在其他模块)

Ⅴ信息的文字反馈

·利用QLineEdit给出耗时和花费的反馈

·利用QTextEdit给出路线、状态的反馈

② 需求处理模块

Ⅰ解决费用最少需求

·算法主体思想：由于不考虑时间因素，可以单纯的用Dijkstra算法，但是考虑到必经城市，我们考虑**采用Floyd算法**，原因如下：

假设城市数为n，选择的必经城市数为m，必经城市数的全排列为m！，那么利用Dijkstra算法，需要运行(m + 1) \* m! = **(m + 1)!次**Dijkstra算法，再选择出一条最省钱的路径，那么算法的时间复杂度为**(m + 1)! \* n²**。若采用Floyd算法，算法复杂度为**n³**,且Floyd算法将任意两地的最短路径都计算出来了，所以考虑必经城市时，只需要简单的拼接再比较。

·算法步骤:

(0)采用**firstAdjMatrix矩阵**，**任意两地之间最多有一条费用最少的车次(航班)**

(1)若选择了必经城市则计算必经城市全排列(递归)

(2)利用Floyd算法，计算出任意两地花费最少的路径

(3)若选择了必经城市则对必经城市全排列的每一种情况进行拼接，选出最佳路径

(4)根据最佳路径算出花费时间，花费金钱已经在选择最佳路径时得出

Ⅱ解决时间最少需求

· 算法主体思想：在解决时间最少问题时，我们曾考虑仿照算法一利用Floyd算法，但是算法二需要考虑到时间因素，并不是各个航班时间的累加，和出发时间有关系，而Floyd算法是用每个点对整个图进行“疏松”操作，而我们现在只掌握了出发点的出发时间(**非**车次(航班)出发时间)，其它点的出发时间尚未掌握，所以不可以使用Floyd算法，于是我们决定采用Dijkstra算法，进行(m + 1)!运算，这也就是为什么要限制选择必经城市的数量

·算法步骤

(0)采用**allAdjMatrix矩阵**，**具有所有车次(航班)的信息**

(1)若选择了必经城市则计算必经城市全排列(递归)

(2)利用Dijkstra算法，计算出指定两地花费时间最少的路径

**其中在利用A地更新其他城市时，需要得出A地到其他城市花费时间最少的车次(航班)，利用此车次(航班)进行比较和更新**

(3)选择出花费时间最短的路径

(4)根据最佳路径算出花费金钱，花费时间已经在选择最佳路径时得出

Ⅲ解决限时时间最少费用需求

· 算法主体思想：此问题是比较灵活的一个问题，给出的限时时间不同，得出的路线可能会不同，我们经过讨论，打算**先调用“最少花费”算法**，如果得出路径的时间小于限时，则最优路径为此路径；如果“最少花费”算法得出的路径的时间大于限时时间，那么**再调用“最少时间”算法**，如果得出的路径的时间大于限时时间，那么肯定没有满足条件的路径，如果得出的路径时间小于限时时间，那么我们**可能**得到了一个花费最少的最优解，我们可以利用其花费，在第三个算法的回溯法中，尽可能的剪枝来缩短搜索范围。

剪枝的条件:

(1)超时剪枝。当路线时间已经超过限制时间，回溯

(2)超费用剪枝。和算法二得出的费用比较，大于则回溯

(3)递归深度剪枝。人为设置递归深度

·算法步骤

(0)采用**allAdjMatrix矩阵**，**具有所有车次(航班)的信息**

(1)调用算法一，判断时间是否小于限制时间，如果小于则结束，不然进入(2)

(2)调用算法二，判断时间是否大于限制时间，如果大于则结束，不然进入(3)

(3)递归搜索，利用上述3种规则回溯

(4)搜索到一条可行路径时，和“当前最优”路径比较，若更优则替换。回溯

Ⅳ解决查询状态需求

·算法思想：此算法相对简单，接收一个用户点击查询按钮时候的时间，从路线的头部开始累加时间，和用户想要查询到的时间相比对，当累计时间大于等于用户点击查询的时间时，就找到了对应状态

③ 日志模块

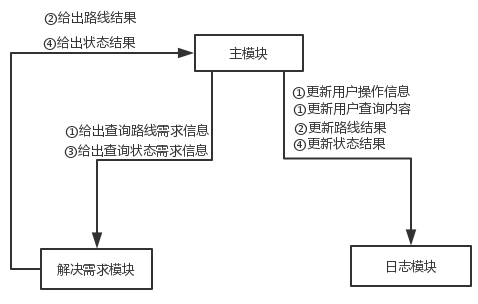
Ⅰ创建文件

以ios::out 和 ios::app 创建文件

Ⅱ更新文件

向外提供接口，接收字符串，追加写入文件

2、模块调用关系

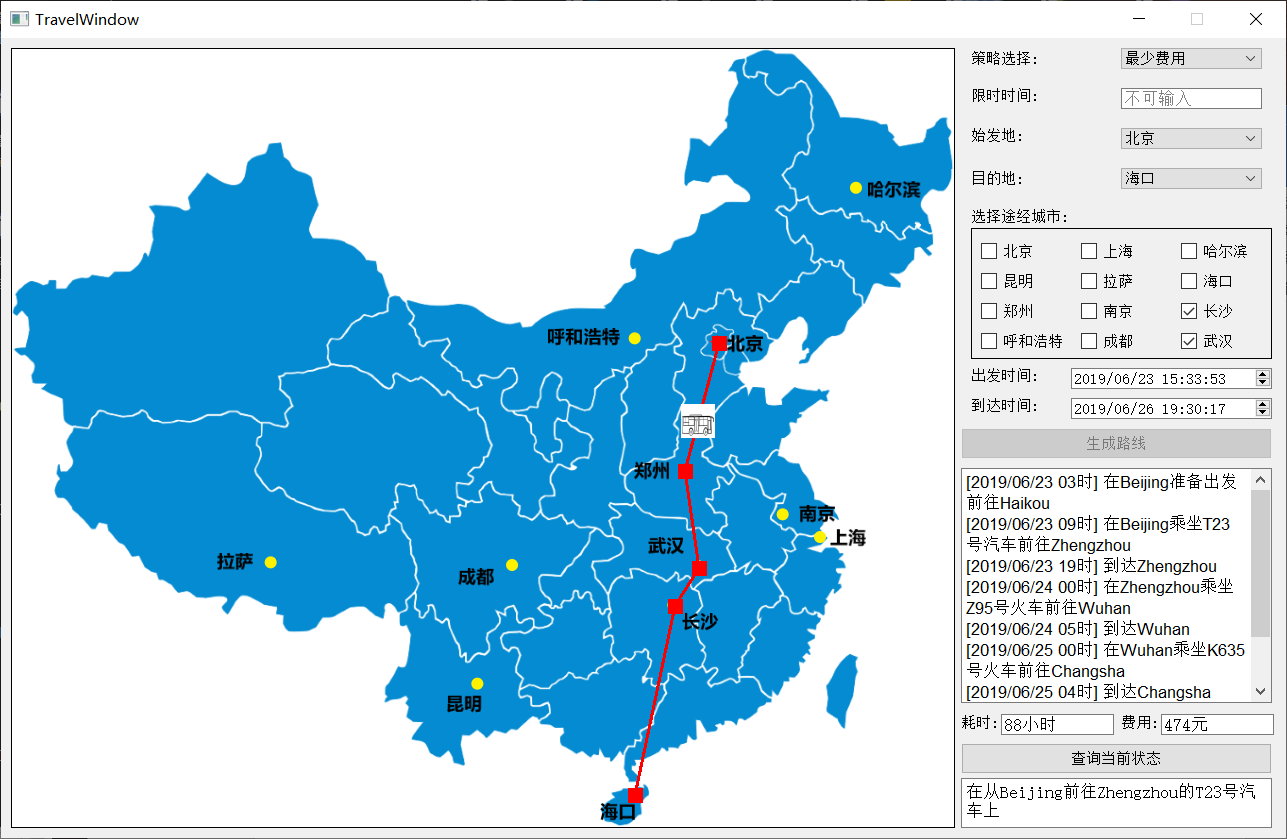


其中序号小的操作需要先完成，序号大的操作才能完成

六、测试样例

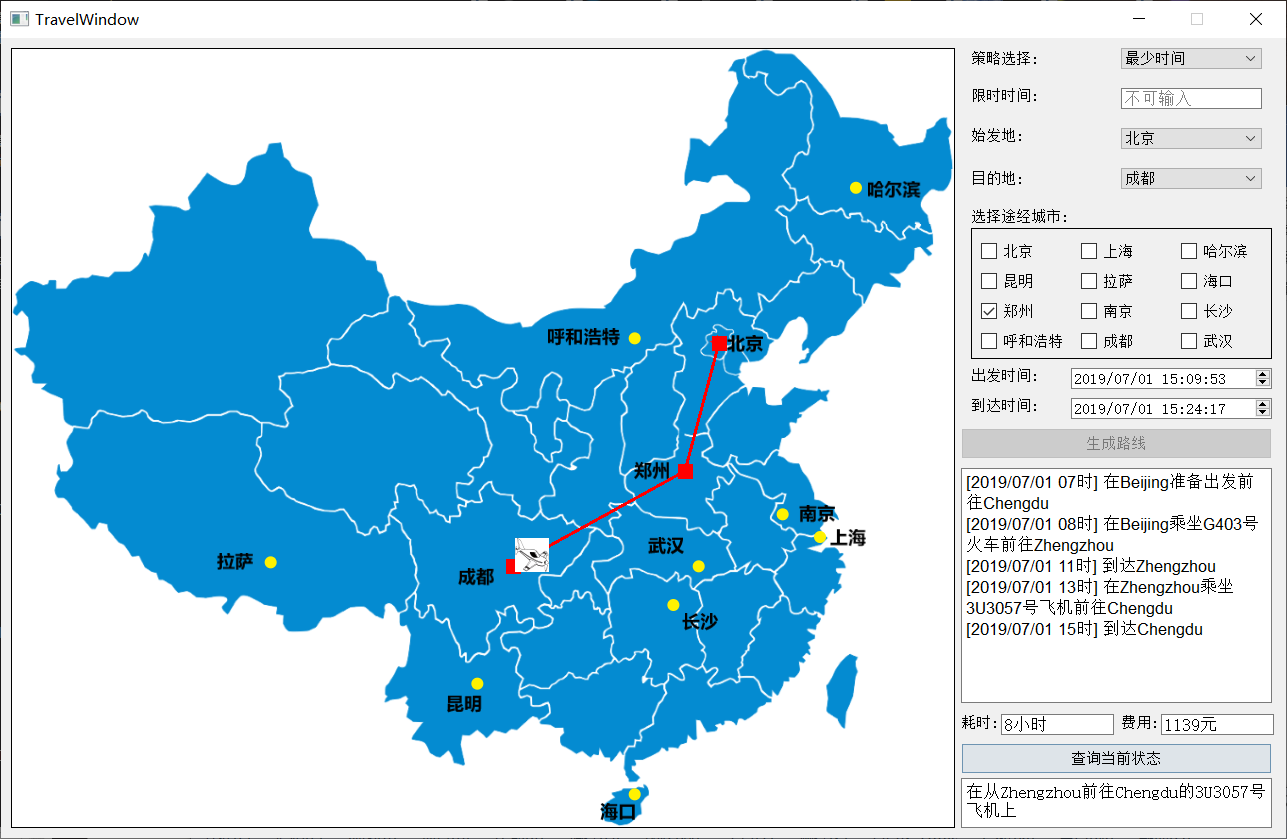
最少费用:

2019/06/23 15:33:53 由北京出发，前往海口，必经长沙、武汉



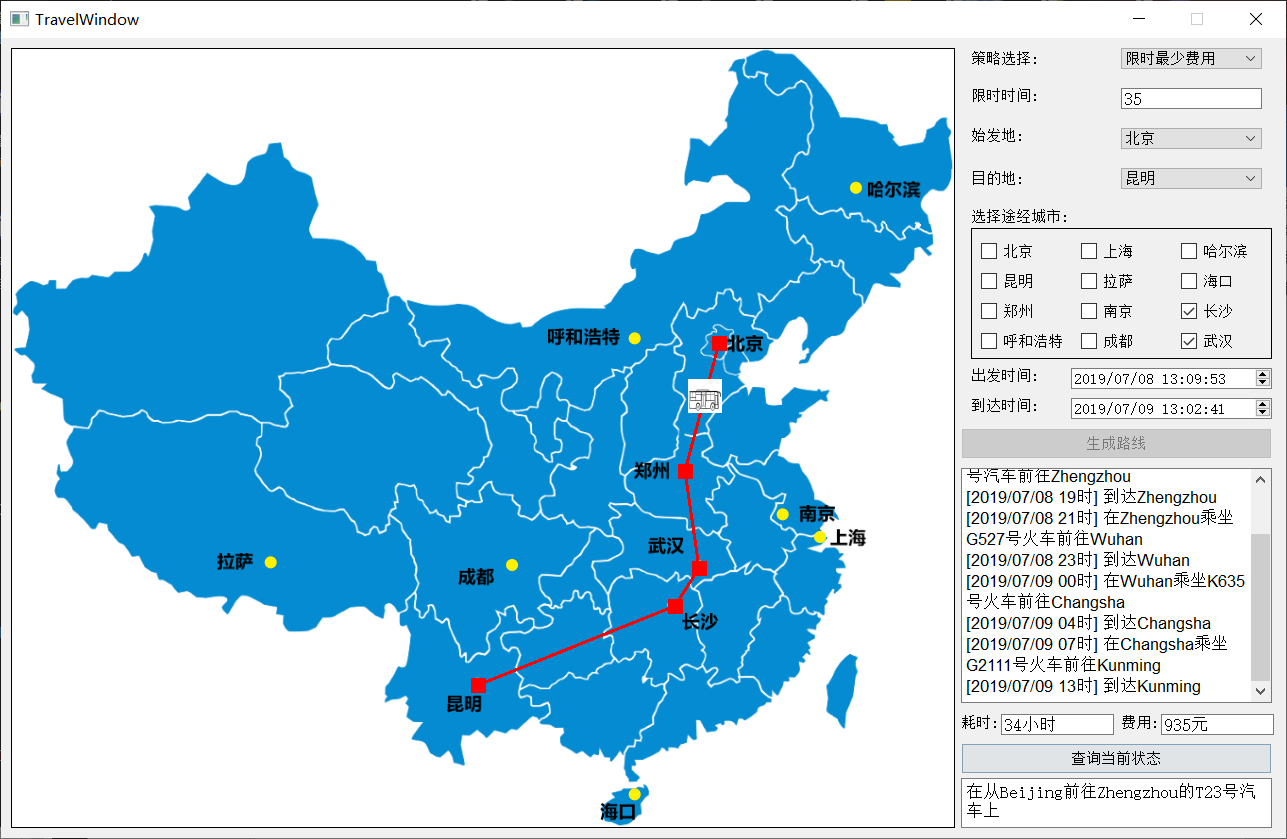
最短时间:

2019/07/01 15:09:53 由北京出发前往成都，必经郑州



限时时间最短时间

2019/07/08 13:09:53 由北京出发前往昆明，必经长沙武汉



日志系统



七、评价和改进意见

1、评价

本组对此项目的完成度较高，实现了基本功能，并完成了图形化的工作，对算法的选用和优化我们都做了很多努力。但是我们同时也认识到了自己的不足，前期工作较为缓慢，导致后期工作紧张。编码时马虎大意，造成了一些不必要的BUG，算法方面知识匮乏，没能把问题解决到最好

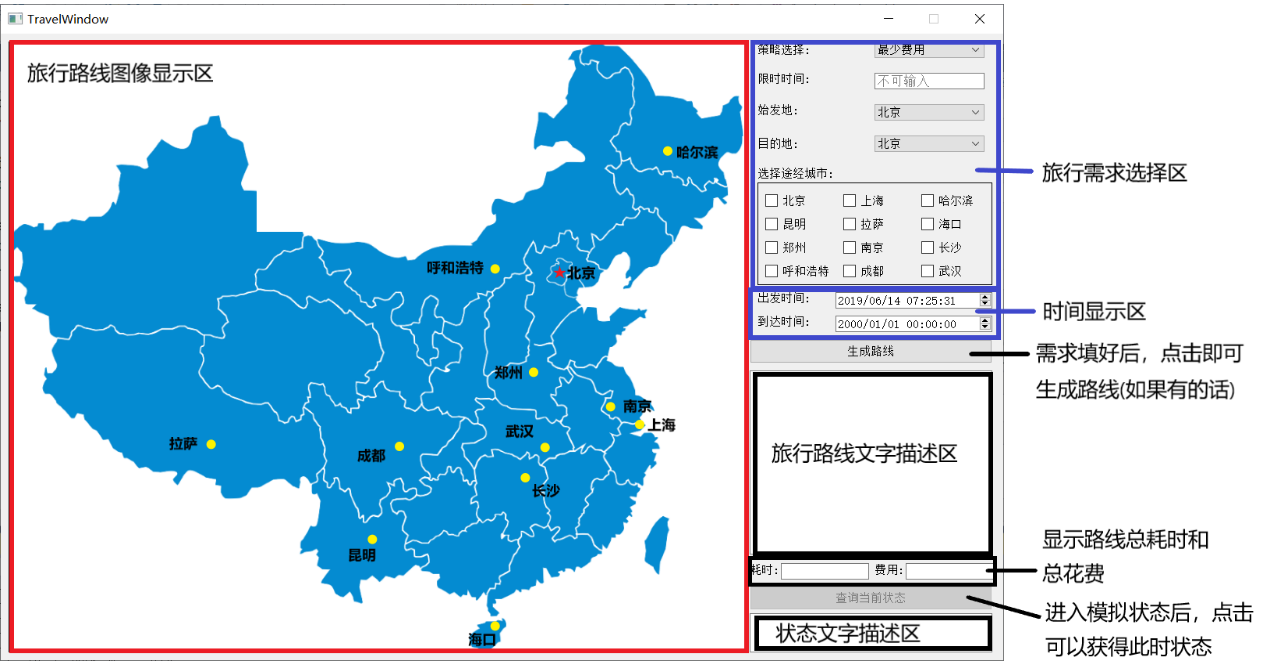
2、改进意见

·解决“最少时间”问题时，多次调用Dijkstra算法，导致一些城市间的最短时间路径多次重复计算，可不可以减少重复的计算？

·解决“限制时间最少花费”问题时，回溯法运行时间过长，有无更好的剪枝算法？

八、用户使用说明

1、各部分功能介绍



2、运行图

