旅行查询与模拟系统

课程设计报告

版本号：2.0

编制时间：2016.06.03

编制人员：杨炫越 张晓宇 刘子欣

1. 设计任务的描述

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）。

1. 功能需求说明及分析
   1. 基本功能需求
      1. 城市总数不少于10个
      2. 建立汽车、火车和飞机的时刻表（航班表），包含沿途到站及票价信息
      3. 旅客的要求包括：起点、终点、途径某些城市和旅行策略
      4. 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：在规定时间内所需费用最少

* + 1. 旅行模拟查询系统已时间为轴向前推移，以10秒左右向前推进1个小时（非查询状态的 请求不计时）；
    2. 不考虑城市内换成交通工具所需时间
    3. 系统时间精确到小时
    4. 建立日志文件，对旅客状态变化和键入等信息进行记录
  1. 拓展功能
     1. 某旅客在旅行途中可更改旅行计划，系统应做相应的操作
     2. 用图形绘制地图，并在地图上反映出旅客的旅行过程

1. 总体方案设计说明
   1. 软件开发环境

**操作系统**：Windows 10 Pro

**IDE**：Code::Blocks 16.01

Qt Creator 3.6.1 Based on Qt 5.6.0

**编译器**：MinGW 4.9.2 32bit

* 1. 总体结构
  2. 模块划分

**启动模块**：main

**数据结构模块**：structs

**方案查询模块**：strategies

**图形化界面模块**：

**主界面模块**：mainWindow

**添加旅客模块**：addDialog

**更改需求模块**：changeDialog

**中心框架模块**：mainWidget

**旅客状态实时显示模块**：infoWidget

**地图与旅行实时模拟模块**：mapWidegt

1. 数据结构说明和数据字典
   1. 数据结构说明
      1. 常量定义

#define INF 0x3f3f3f3f //无穷大

#define STRATEGY\_NUM 3 //策略种数

enum Strategy //策略

{

MIN\_COST, //最小费用

MIN\_DUR, //最短时间

LIM\_DUR\_MIN\_COST //限定时间最小费用

};

#define MAX\_EDGE\_NUM 10000 //最大边数量

#define VEHICLE\_KIND\_NUM 4 //交通工具种数

#define CITY\_NUM 10 //城市数量

#define ONE\_HR 1 //一小时

#define HR\_PER\_DAY 24 //每天小时数

#define FREE 0 //免费

//第三种策略的A\*搜索中，

//当最小费用路径的总时间超过旅行时间限制，

//且差达到此阈值时，采用以时间为主的搜索方式

#define DUR\_DIFF\_THRESHOLD 20

//第三种策略的A\*搜索中，

//当必须途经城市数量不超过CITY\_NUM\_THRESHOL时，

//且时间限制不超过DUR\_LIM\_THRESHOLD时，

//采用以时间为主的搜索方式

#define CITY\_NUM\_THRESHOLD 3

#define DUR\_LIM\_THRESHOLD 40

* + 1. 全局变量定义

//当前城市，用于对A\*搜索中结点访问优先次序作判断

int curCity;

//旅客数组

vector<Passenger> passengers;

//城市名数组

string TrafficMap::cityNoToName[CITY\_NUM] =

{

"北京",

"长春",

"石家庄",

"西安",

"南京",

"上海",

"广州",

"长沙",

"成都",

"重庆"

};

//当前所选中查看状态的旅客

int selectedPassenger = -1;

//交通路线图

TrafficMap trafficMap;

* 1. 数据字典

struct Passenger //旅客结构体

{

struct Demand //旅客需求结构体

{

string name; //旅客姓名

string startCity, destCity; //出发地与目的地

vector<string> passCities; //沿途必经城市

int startTime; //出发时刻（小时）

QDateTime startDateTime; //出发日期

Strategy strategy; //旅行策略

int durLimit = INF; //旅行时间限制

} demand;

struct ChangedDemand //需求更改结构体

{

bool changeFlag = false; //标记是否更改了需求

string startCity, destCity; //出发地与目的地

vector<string> passCities; //沿途必经城市

int startTime; //出发时刻（小时）

QDateTime startDateTime; //出发日期

Strategy strategy; //旅行策略

int durLimit = INF; //旅行时间限制

} changedDemand;

struct State //旅客状态结构体

{

int totDur; //旅行总时间

int totCost; //旅行总费用

vector<int> revPath; //旅行路径（逆向存储）

QPropertyAnimation \*anime; //旅行演示动画

QLabel \*label; //旅客显示图片

int currentVehicleKind; //当前交通工具种类

int remainSlot; //剩余时间槽数

int nextEdgeIndex; //下一条边索引

} state;

void changeDemand(void); //需求更改函数

bool del = false; //标记该旅客是否已被删除

};

struct TrafficMap //交通路线图结构体

{

#define MAX\_EDGE\_NUM 10000 //最大边数量

#define VEHICLE\_KIND\_NUM 4 //交通工具种数

enum VehicleKind //交通工具种类

{

//巴士，火车，飞机，缺省（即在一个城市候车）

BUS, TRAIN, PLANE, DEFAULT

};

#define CITY\_NUM 10 //城市数量

#define ONE\_HR 1 //一小时

#define HR\_PER\_DAY 24 //每天小时数

#define FREE 0 //免费

static string cityNoToName[CITY\_NUM]; //城市名数组

map<string, int> cityNameToNo; //城市名转城市编号

struct Coor //坐标结构体

{

int x;

int y;

} cityCoor[CITY\_NUM] = //城市坐标数组

{

{531, 224},

{626, 151},

{513, 259},

{436, 330},

{578, 349},

{615, 360},

{510, 498},

{498, 424},

{376, 389},

{417, 404}

};

struct CityTime //城市与时刻结构体（用于表示状态）

{

int city, time;

};

struct Edge //边结构体

{

VehicleKind kind; //交通工具种类

string vehicleNo; //车次/航班号

CityTime depart, arrive; //出发城市与时刻，到达城市与时刻

int dur, cost; //时间与费用

bool del; //标记该边是否被删除

bool vis; //标记该边所达状态是否已被访问

} edges[MAX\_EDGE\_NUM];

int edgeNum; //边数量

//交通路线表（邻接表形式），存储从某个城市某个时间出发的所有交通路线

vector<int> vehicleTable[CITY\_NUM][HR\_PER\_DAY][VEHICLE\_KIND\_NUM];

//任意两个城市在任意两个时刻之间的最短费用

int minCost[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM][HR\_PER\_DAY],

//任意两个城市在任意两个时刻之间的最短时间

minDur[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM][HR\_PER\_DAY];

//任意城市任意时刻出发到另一个城市最小费用的到达时刻

int minCostArriveTime[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM],

//任意城市任意时刻出发到另一个城市最短时间的到达时刻

minDurArriveTime[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM];

//任意两个城市在任意两个时刻之间最小费用路径的前驱边（用于路径还原）

int minCostPrv[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM][HR\_PER\_DAY],

//任意两个城市在任意两个时刻之间最短时间路径的前驱边（用于路径还原）

minDurPrv[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM][HR\_PER\_DAY];

//任意两个城市在任意两个时刻之间最小费用路径的途经城市（位压缩存储）

int minCostPass[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM][HR\_PER\_DAY],

//任意两个城市在任意两个时刻之间最短时间路径的途经城市（位压缩存储）

minDurPass[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM][HR\_PER\_DAY];

//任意两个城市在任意两个时刻之间最小费用路径的总时间

int minCostDur[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM][HR\_PER\_DAY],

//任意两个城市在任意两个时刻之间最短时间路径的总费用

minDurCost[CITY\_NUM][HR\_PER\_DAY][CITY\_NUM][HR\_PER\_DAY];

//构造函数

TrafficMap()

//加边函数

void addEdge(VehicleKind \_kind, const string& \_vehicleNo,

int \_departCity, int \_departTime,

int \_arriveCity, int \_arriveTime,

int \_dur, int \_cost)

//删边函数

void eraseEdge();

//从文件中读取路线表

void readFile();

deque<CityTime> q; //用于SPFA最短路算法的队列

//SPFA最短路算法中标记某个状态（城市，时刻）是否已被访问

bool vis[CITY\_NUM][HR\_PER\_DAY];

//SPFA最短路算法，预处理出上述所有涉及两节点间

//最小权值路径的数组

void SPFA();

struct HeapNode //A\*搜索中的估价函数结点

{

//该结点在入堆的序号，用于路径还原

int saveId;

//该结点所在城市与时刻

int city, time;

//A\*搜索过程估价函数构成：

//f = g + h

//g为起始节点到当前结点路径的权值

//h为启发式函数，使用预处理的当前结点到目标节点的最小权值

//f为估价函数，估计在当前节点状态下

//到目标节点最小路径总权值

int g, f;

//gPass为到当前节点已经过的城市（位压缩存储）

//fPass为若走预估最小路径，到目标节点将途径的城市（位压缩存储）

int gPass, fPass;

//到当前节点路径的时间与费用，用于第三种策略

int dur, cost;

//在A\*搜索中，进行BFS时优先拓展估价函数值小的结点

//当估价函数值相同时，优先拓展当前已走路径权值小的结点

//再则优先拓展能离开当前城市的结点（防止在同一个城市死循环）

//再则优先拓展当前已走路径时间短的结点（用于策略三）

bool operator >(const HeapNode& rhs) const;

//用于getStrategy()中A\*搜索的最小堆（优先队列）

priority\_queue<HeapNode, vector<HeapNode>, greater<HeapNode>>

heap;

//保存A\*搜索中已入堆结点的前驱信息的结构体，用于路径还原

struct Save

{

//到达该结点的边

int prvEdge;

//该结点的前驱结点

int prv;

};

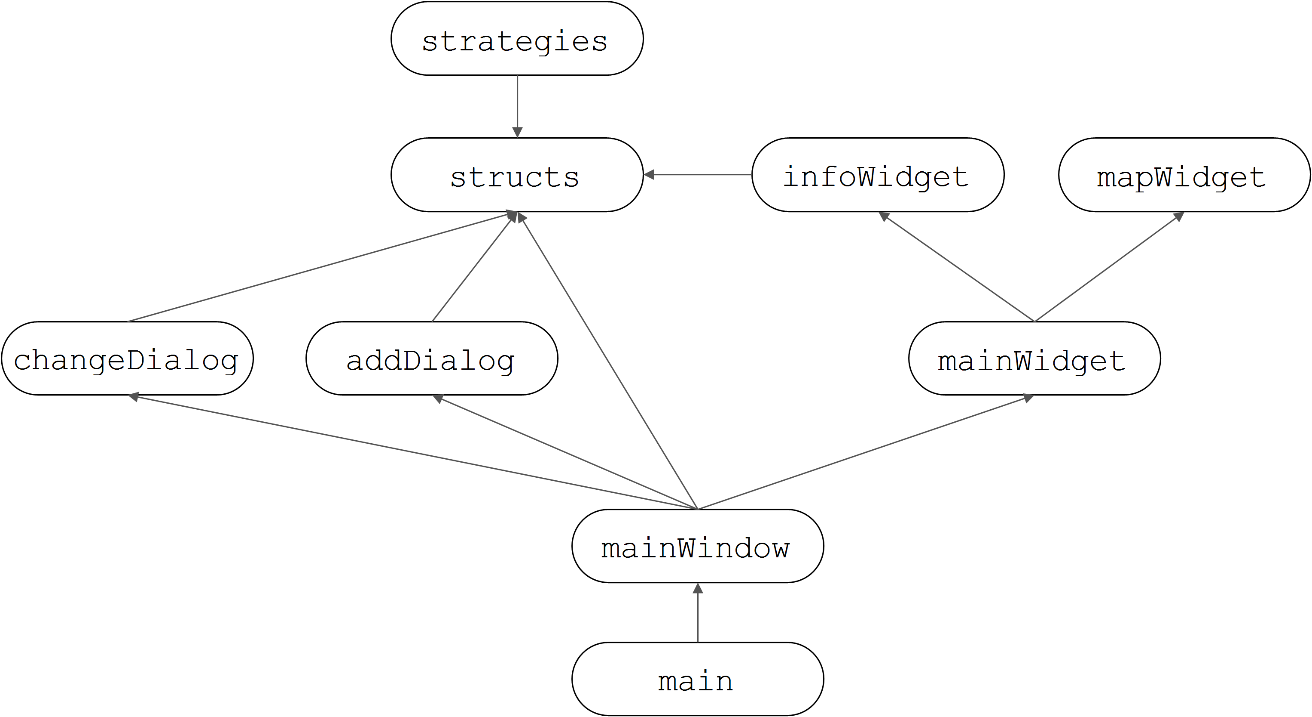
//传入一个旅客结构体，分析其需求，获取旅行方案

void getStrategy(Passenger& passenger);

//用于A\*搜索后边的标记状态的还原

void restoreEdge();

};

1. 各模块设计说明
   1. 系统模块划分
      1. 模块关系图
      2. 模块具体描述
         1. **main**

本模块创建了图形界面的主显示窗口，并对显示参数进行初始化配置，然后在运行期显示该窗口。

* + - 1. **structs**

本模块定义了旅客结构体与交通路线图结构体。

旅客结构体中将旅客细分为需求与状态两个部分：

**需求部分**用于记录旅客的旅行需求：出发地与目的地，出发日期与时刻，必须途径的城市，及最小费用、最短时间、限时最小费用三种旅行策略。

**状态部分**用于存储该旅客已通过查询系统获得的旅行方案，以及该旅客的实时状态、显示动画等。

交通路线图结构体是该系统的核心，以邻接表的形式存储了汽车、火车及飞机三种交通工具的时刻表，并对外提供查询旅行方案的接口。该结构体设计中为提高查询系统搜索路线的效率做出了如下两点优化：

1. 将每个城市拆分成24个节点，代表该城市在每天每个时刻的状态。建图过程如下：

对同一城市的相邻时刻，连一条时间1个小时，费用为0的边；

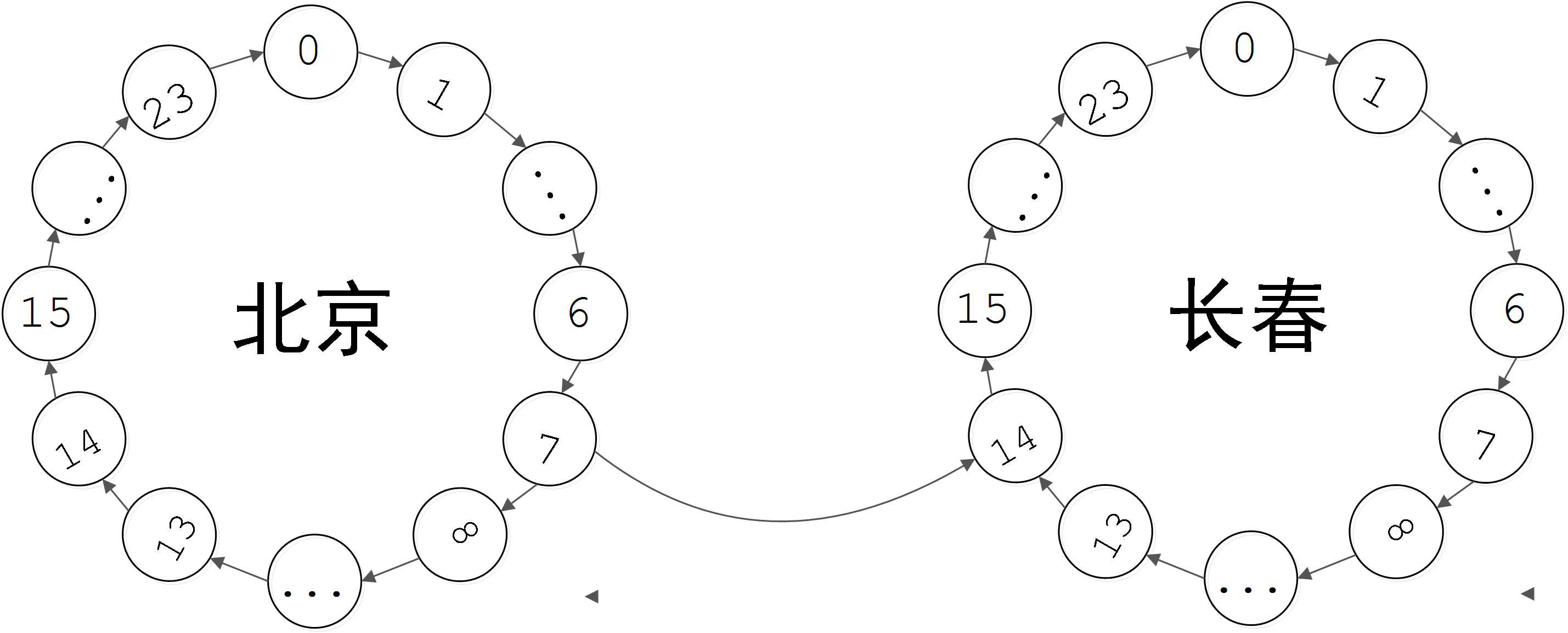
对一条航线或一班车次，从出发城市与时刻到到达城市与时刻连一条对应时间与费用的边。

以火车班次T1（北京7:00-长春14:00，费用267元）为例，连边如下：

种类：火车

费用：267元

时间：7小时



此种方式避免了城市与时刻的耦合，使得在搜索过程中不必查找下一班车次的出发时间（若需等待，只需在同一城市走向下一时间结点即可），从而将最短时间与最小费用的搜索过程统一起来，对以时间为权值的路径无须作任何特殊处理。并且，此种建图方式极大方便了对搜索过程中以各种最短路径作为启发式函数的预处理，见2）。

2）预处理了一系列最短路径有关的数组，如任意两个城市任意两个时刻之间最短路径权值、前驱边、途径城市等。其中权值分为费用与时间，以1）所述方式建图后，这两者可以相互独立出来进行处理。这些预处理结果一是可作为对带约束最短路径进行A\*搜索（见5.1.2.3.）的启发式函数，二是当从预处理结果可判断出已符合旅客需求（如沿途必经城市）时，可立即结束搜索，并进行路径还原得到旅行方案。经实践，该预处理过程显著加速了搜索过程。

* + - 1. **strategies**

本模块实现了**structs**模块中提供查询旅行方案的接口getStrategy。该接口原型为

void getStrategy(Passenger& passenger)，

其中passenger是一个旅客类型的引用。传入该参数后，该接口分析这个旅客的需求passenger.demand，并将其转化为一个图上带约束的最短路径问题，约束为必经点集的约束（必须途径的城市）与权值的约束（限定时间）。求解该问题后，该接口将所得的最短路径（即旅行方案）写入到该旅客的状态passenger.state中，作为该旅客旅行模拟的路线。在该模块中解决此种问题主要采用了**A\*搜索算法**。

A\*搜索算法是一种静态路网中求解[最短路径](http://baike.baidu.com/view/349189.htm)最有效的直接搜索方法，它与一般搜索算法不同点在与它为每一个即将访问的结点计算一个**估价函数**，其构成如下：

，

其中：

为**起始节点到当前结点路径的权值**；

为启发式函数，预估**了当前结点到目标节点最短路径的权值**，当该预估值越接近真实值时，搜索效率越高。而在该问题中，我们可使用经预处理得到的任意两个结点间最短路径权值数组在时间得到一个准确的启发式函数值，从而大大的地提高搜索效率；

为估价函数，估计在当前节点状态下到目标节点最小的路径总权值，反映了**在已到达当前结点的条件下，到达目标节点的最优路径**。

于是在有了每一个待访问结点估价函数值后，A\*搜索的思想即**优先拓展估价函数值最小的结点，此种方式为搜索过程指明了方向，从而避免盲目搜索**。若仅是简单的两点间最短路，利用预处理得到的信息立即就能得到最优解并进行路径还原；而当带上了约束，就需利用预处理信息对该最短路径是否满足约束条件进行判断。而A\*搜索具有一个性质：**搜索结果将按路径总权值递增的次序依次访问目标节点**，此由按估价函数从小至大的次序拓展结点的搜索过程使然。于是便可断定：**第一次得到的满足约束条件的路径必然是最优路径**，从而该搜索算法的**高效性**即**正确性**都得到了保证。

该算法的实现使用了**优先队列**这一数据结构来存储待拓展结点的估价函数值及获取当前队列中的最小估价函数值。优先队列的底层实现基于**小根堆**，故能在时间内快速地获得下一个应该拓展的结点。对第一、二种策略（最小费用、最短时间），搜索算法简述如下：

将起始节点压入优先队列q，其g值为0，h值为到目标节点的最短路径值

while (q非空)

{

当前结点cur = q队首（当前q内估价函数值最小结点）

弹出q队首

if (在已到cur的状态下到目标节点的最短路径满足约束条件)

{

路径还原并记录在旅客状态passenger.state中

return;

}

for (cur的邻接结点)

计算其估价函数值并压入q

}

由于已使用**位压缩**方法存储了任意两个结点间最短路径的途经城市（如用二进制数001011代表途径了第0、1、3个城市），故在判断是否满足约束条件（必须途径的城市）时仅用**位运算**便可快速完成，从而提升了效率。

而对第三种策略（限时最小费用），在此搜索算法框架下有两种不同的思路：

1. **基于最小费用**进行搜索，找到一个总时间时限内的解，**对时限较接近于最小费用路径用时的情况效率较高**；
2. **基于最短时间**进行搜索，找到最短总时间到时限内所有方案的最小费用，**对时限较接近于最短时间的情况效率较高，并可用来判断是否有合法解**。

故在策略三的搜索过程中，**对时限与最小费用路径用时、最短时间的差设定了阈值**，从而能根据不同情况使用相应的搜索策略，使得搜索效率达到最优。

该模块经多次测试，策略一、二在极端情况下的求解平均时间在**4~16毫秒**内，策略三在极端情况下的求解平均时间在**25~40毫秒**，可见该搜索算法**相当高效**。

* + - 1. **mainWindow**

本模块实现了图形用户界面的主界面，包括中心框架mainWidget以及菜单栏menuBar，对menuBar的各个action进行了信号槽的链接。具体的action有:

**添加旅客addAction**：点击将调用addDialog，并对用户的输入进行判定，如果符合条件则成功添加该旅客。

**更改旅客changeAction**：点击将调用changeDialog，并对用户的输入进行判定，如果符合条件则成功更改选中的旅客。

**删除旅客deleteAction**：点击将删除选中的旅客

**显示日志logAction**：点击将暂停系统，弹出日志文件供用户参阅，关闭文件时将会继续系统运行。

**关于aboutAction**：点击将弹出本软件的相关信息。

* + - 1. **addDialog**

本模块实现了添加旅客窗口的图形化Dialog。用户通过输入栏、复选框以及下拉框等控件输入旅客旅行信息，点击确定将会将信息传回MainWindow。具体函数有：

//此函数为AddDialog类的构造函数

AddDialog(QWidget \*parent = 0)

//此函数通过传引用的psg乘客对象，将用户输入的旅客需求返回给//MainWindow

getData(Passenger& psg)

* + - 1. **changeDialog**

本模块实现了更改旅客窗口的图形化Dialog。用户通过输入栏、复选框以及下拉框等控件输入旅客旅行信息，点击确定将会将信息传回MainWindow。具体函数有：

//此函数为ChangeDialog类的构造函数

ChangeDialog(QWidget \*parent = 0)

//此函数通过传引用的psg乘客对象，将用户输入的旅客需求更改返回给//MainWindow

getData(Passenger& psg)

* + - 1. **mainWidget**

本模块实现了对中心框架的划分，还起到了上层信号到下层槽之间的传递作用。具体含有两个控件：

//具体功能见下

MapWidget \*mapWidget

InfoWidget \*infoWidget

具体函数有：

//此函数为MainWidget类的构造函数

MainWidget(QWidget\* parent = 0)

* + - 1. **infoWidget**

本模块实现了旅客用户的实时状态显示，具体包含三个控件：

//此控件实时显示系统当前时间

QLCDNumber \*timeLCDNumber

//此控件实时显示系统当前日期

QLCDNumber \*dateLCDNumber

//此控件实时显示当前在旅行系统中的旅客

QListWidget \*passengerListWidget

//此控件实时显示passengerListWidget中选中的旅客的当前信息

QLabel \*infoLabel

具体函数有：

//此函数为InfoWidget类的构造函数

InfoWidget(QWidget \*parent = 0)

* + - 1. **mapWidget**

本模块实现了旅行系统的实时模拟功能，能够将旅客的模拟线路实时在中国地图上显示出来。具体函数有：

//此函数为MapWidget类的构造函数

MapWidget(QWidget\* parent = 0)

//此函数实现地图的绘制

void draw(QPainter \*painter)

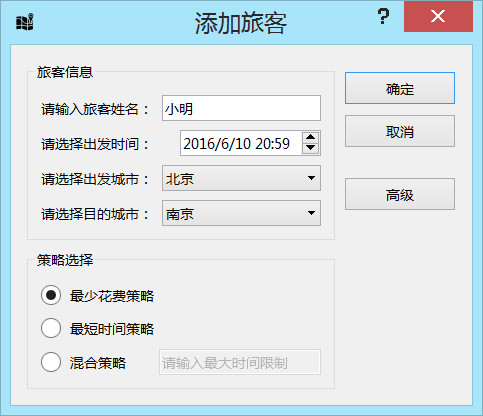
//此函数实现旅行路线的绘制

void drawEdge(QPainter \*painter)

//此函数实现旅行途经城市的绘制

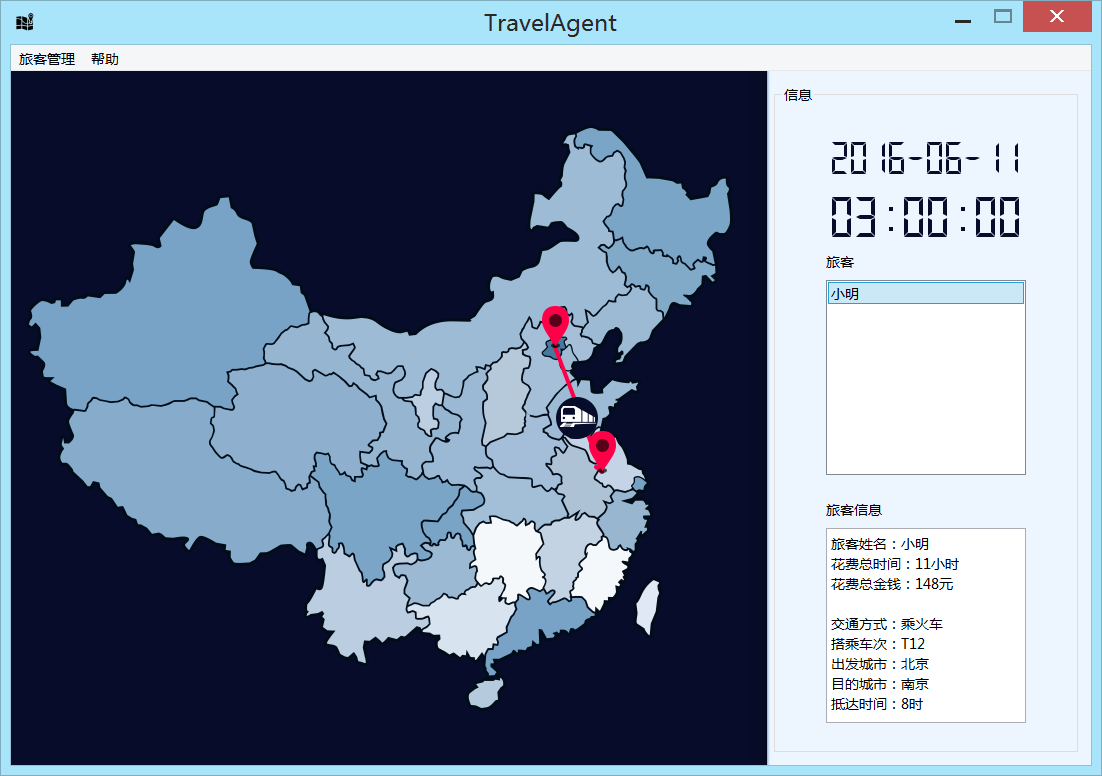
void drawCity(QPainter \*painter)

1. 范例执行结果及测试情况说明
   1. 测试样例1：

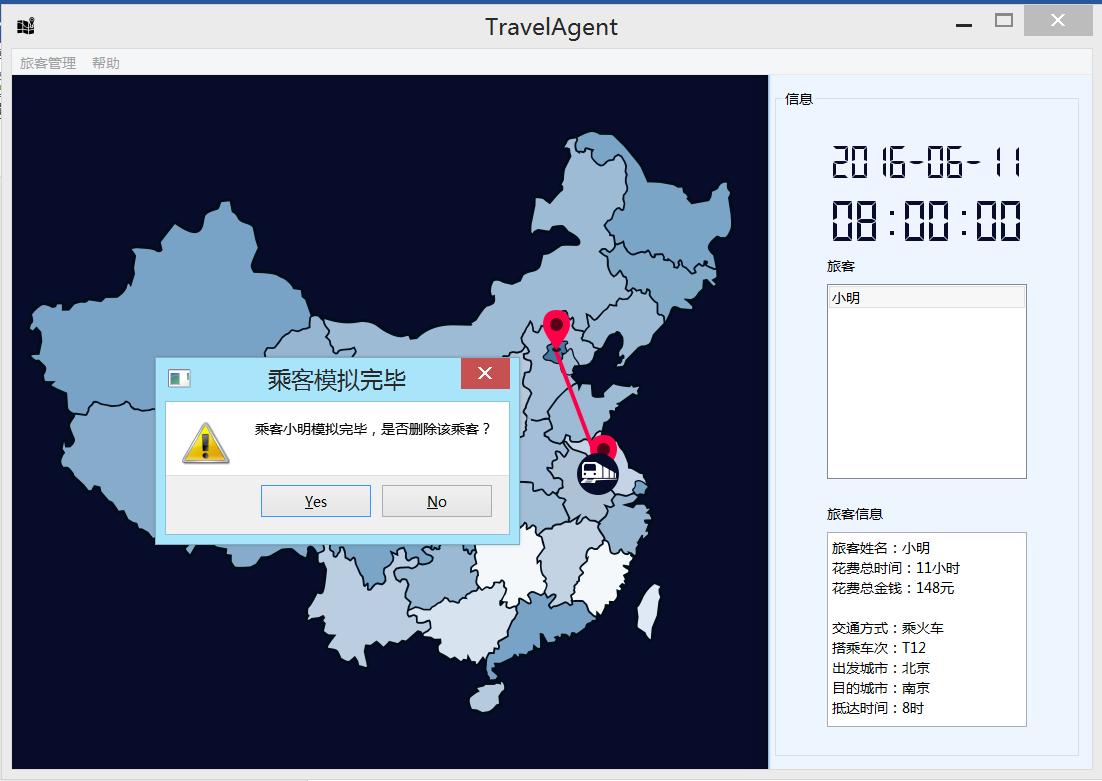


**测试点**：增删旅客、最小费用策略

**测试结果**：

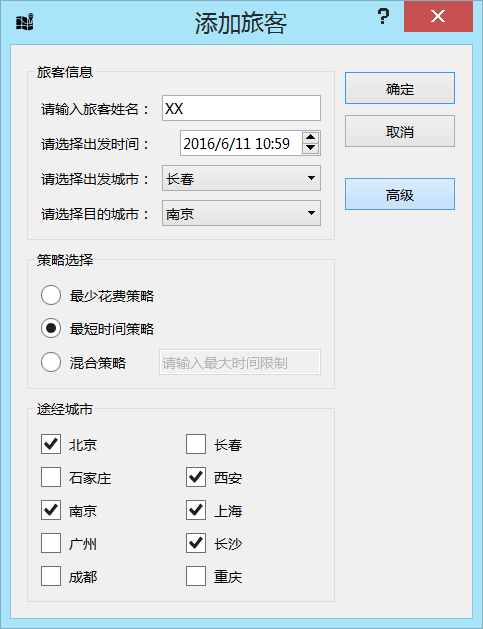


**结果正确**



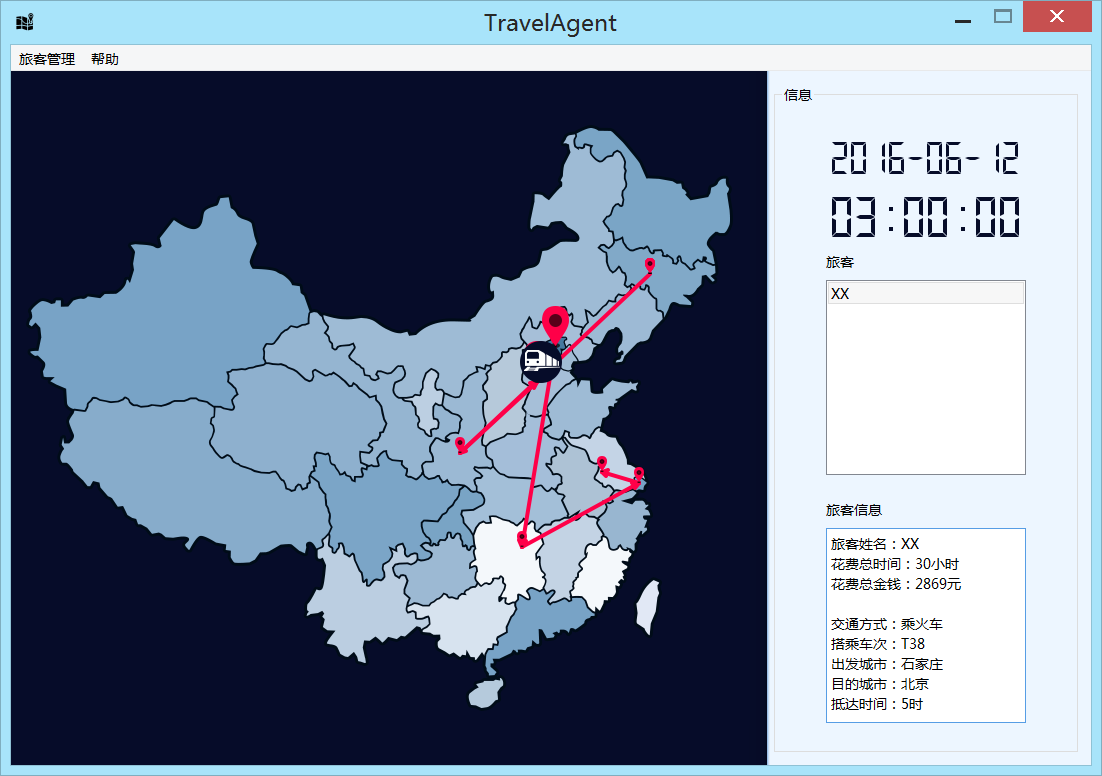
**删除成功**

* 1. 测试样例2：



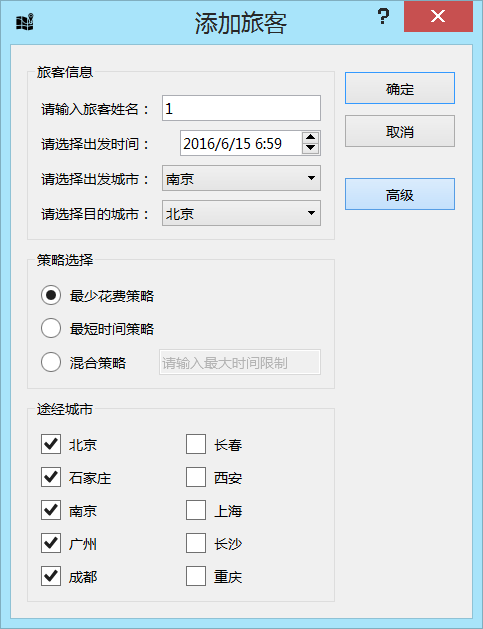
**测试点**：经过固定城市的最短时间策略

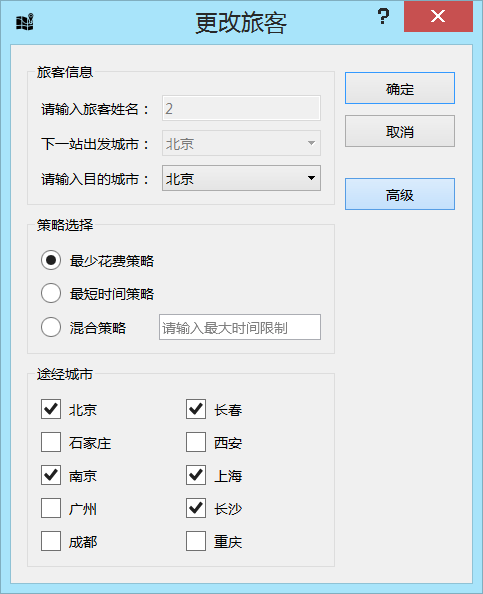
**测试结果**：



**结果正确**

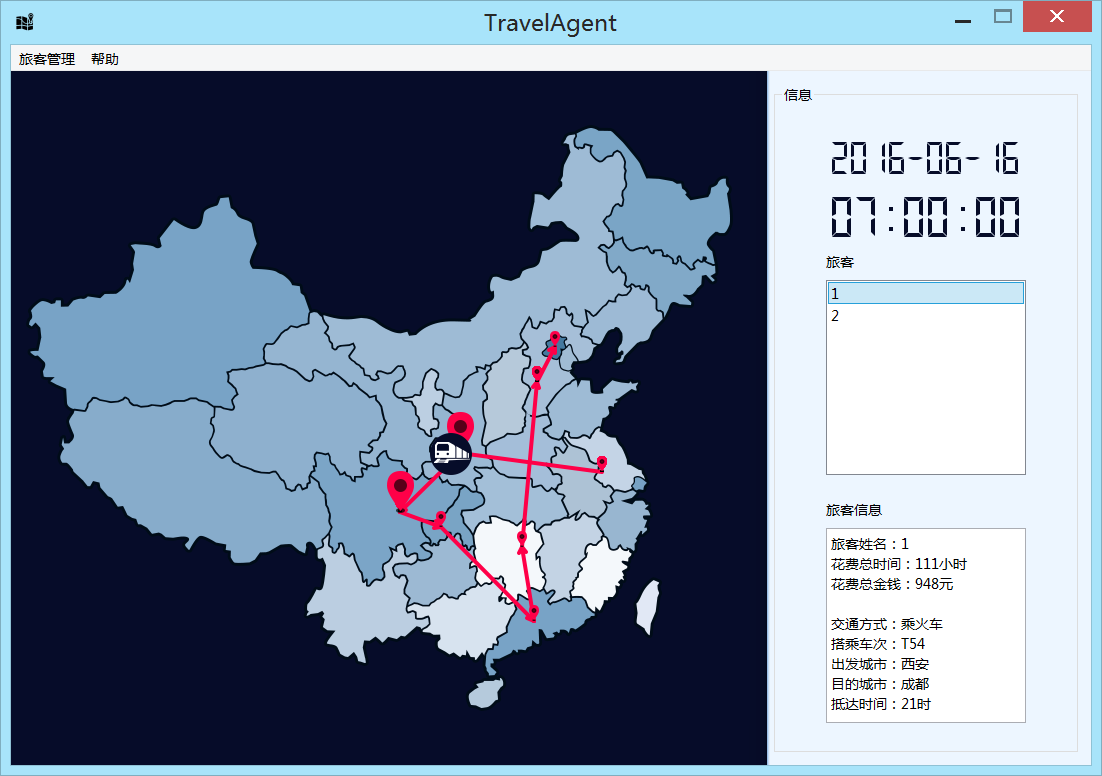
* 1. 测试样例3：

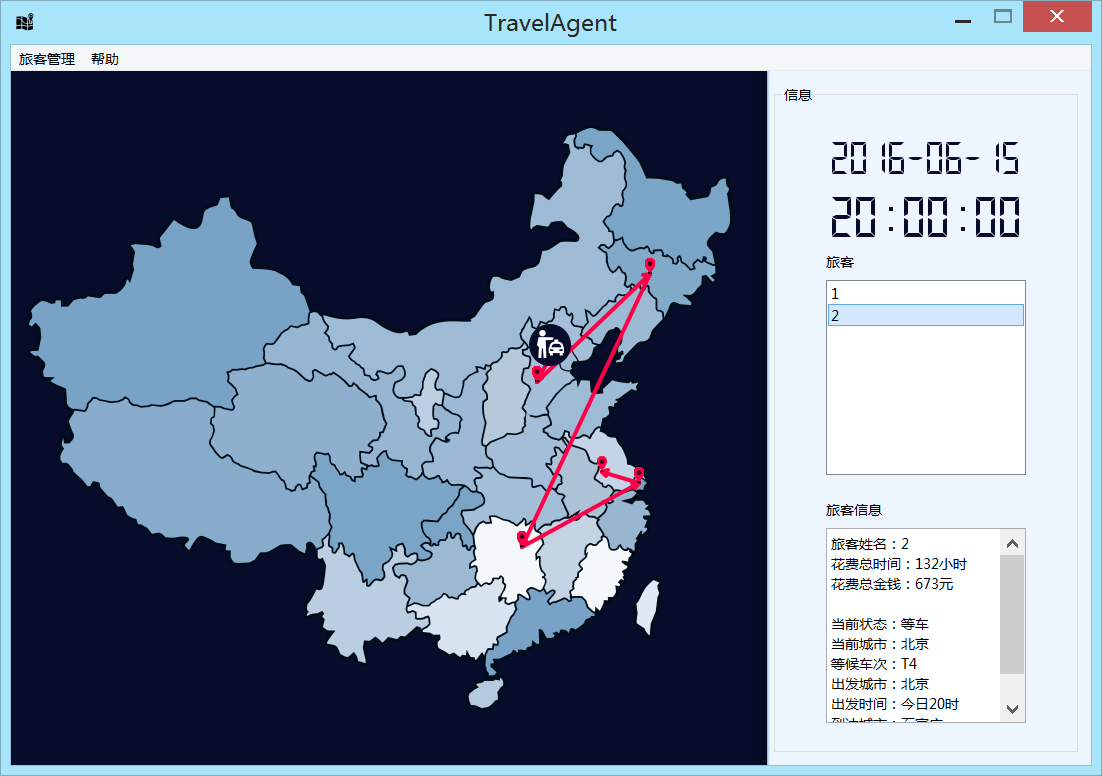




**测试点**：同时添加多个旅客

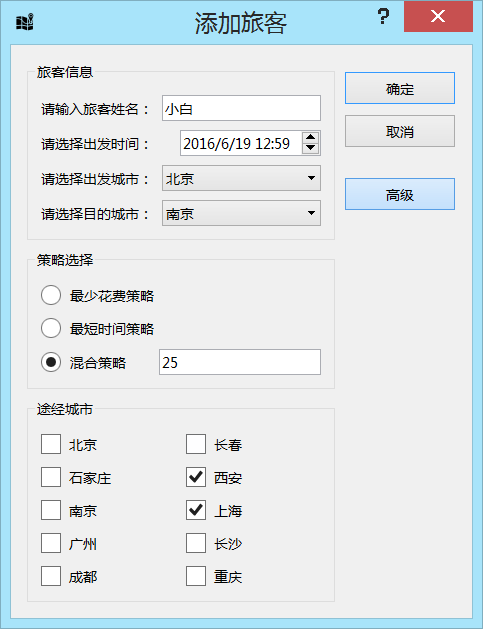
**测试结果**：





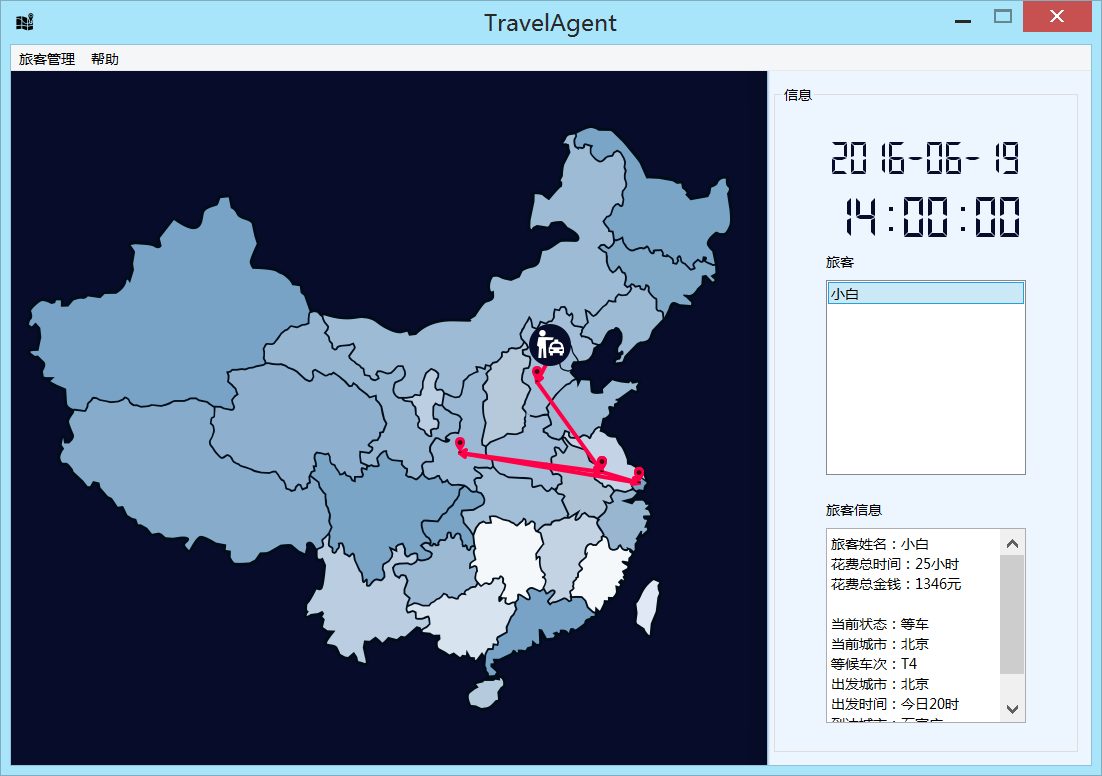
**结果正确**

* 1. 测试样例4：



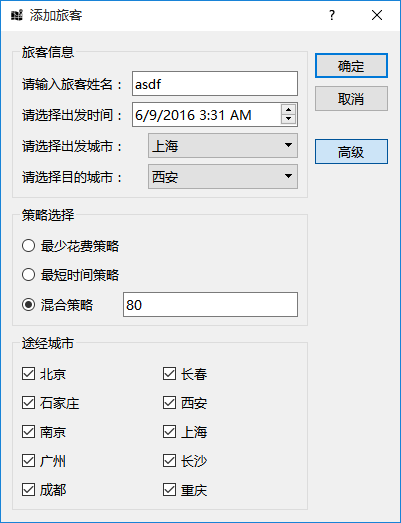
**测试点**：有必须途径的城市及限定时间最小费用策略

**测试结果**：



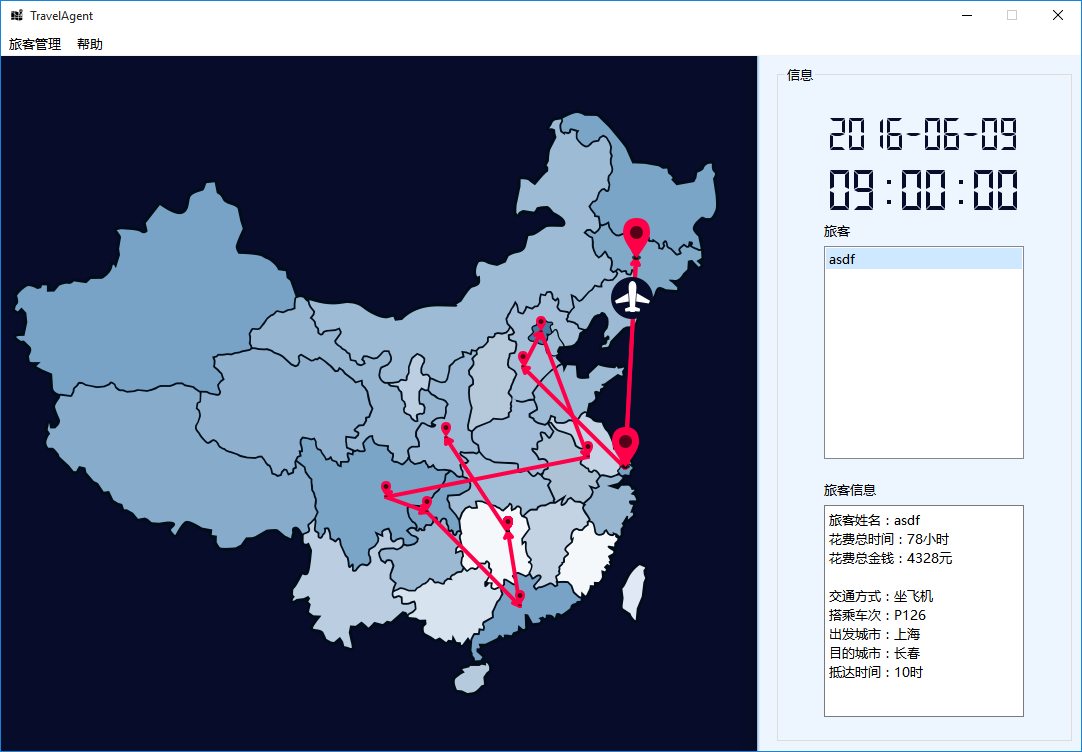
**结果正确**

* 1. 测试样例5：



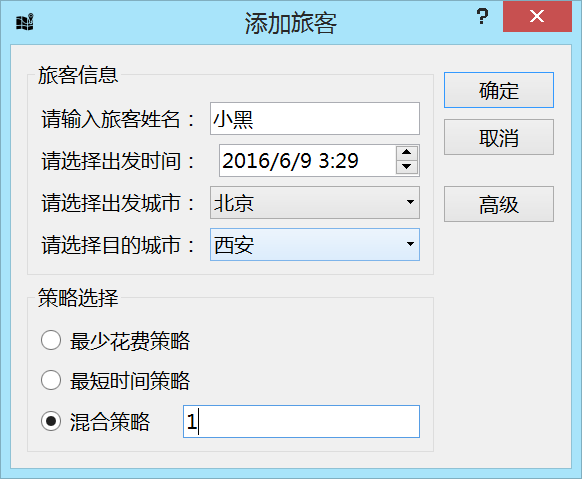
**测试点**：必须途径**所有**城市及限定时间最小费用策略（极端情况）

**测试结果**：



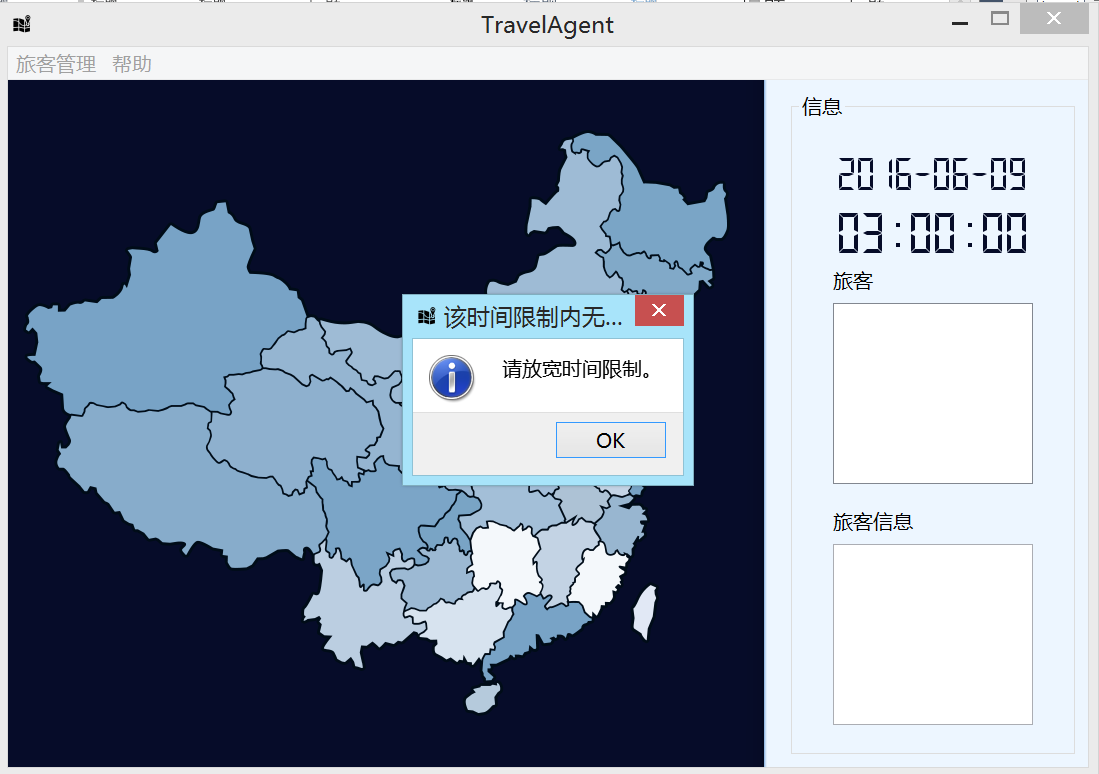
**结果正确**

* 1. 测试样例6：



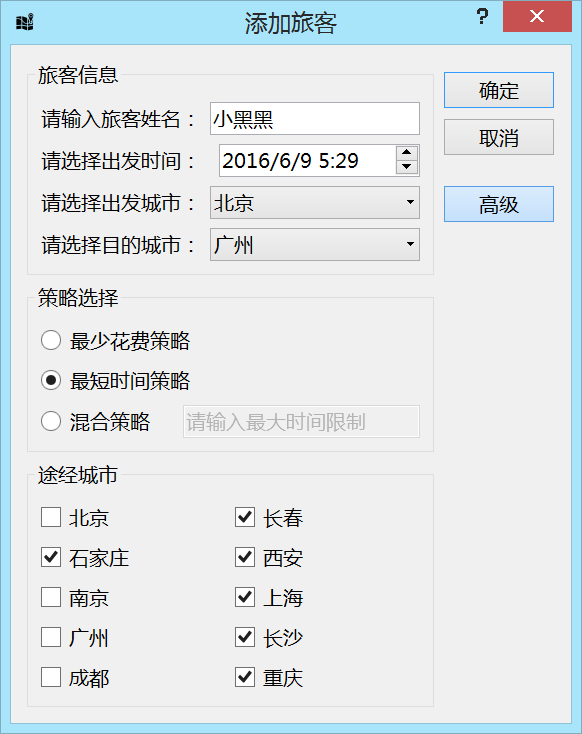
**测试点**：规定时间内无法完成旅行

**测试结果**：



**结果正确**

* 1. 测试样例7：

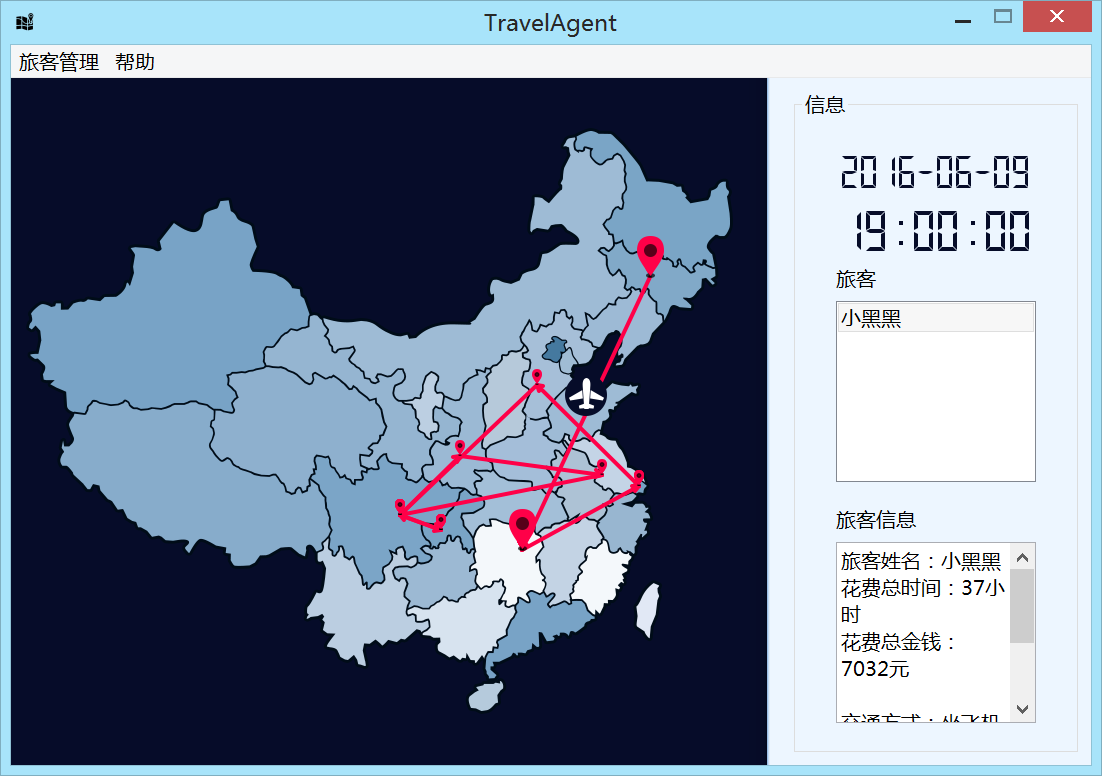


**测试点**：中途更改策略

**测试结果**：

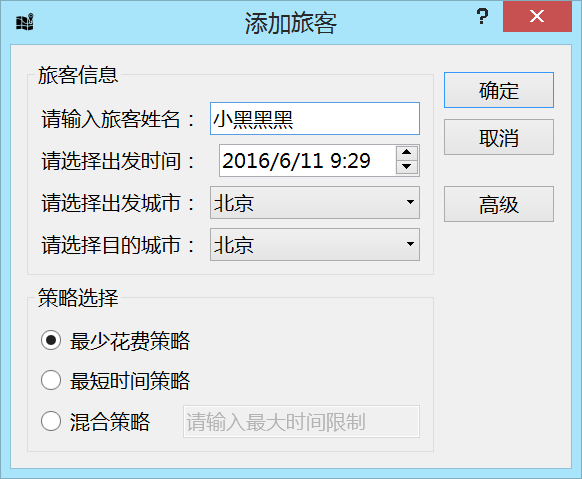






**结果正确**

* 1. 测试样例8：



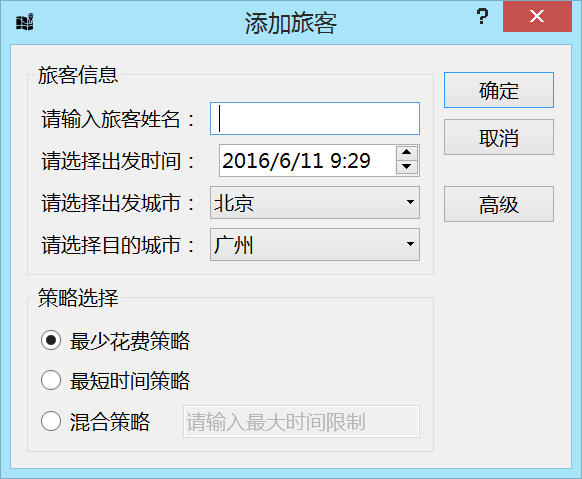
**测试点**：出发地与起始地相同

**测试结果**：



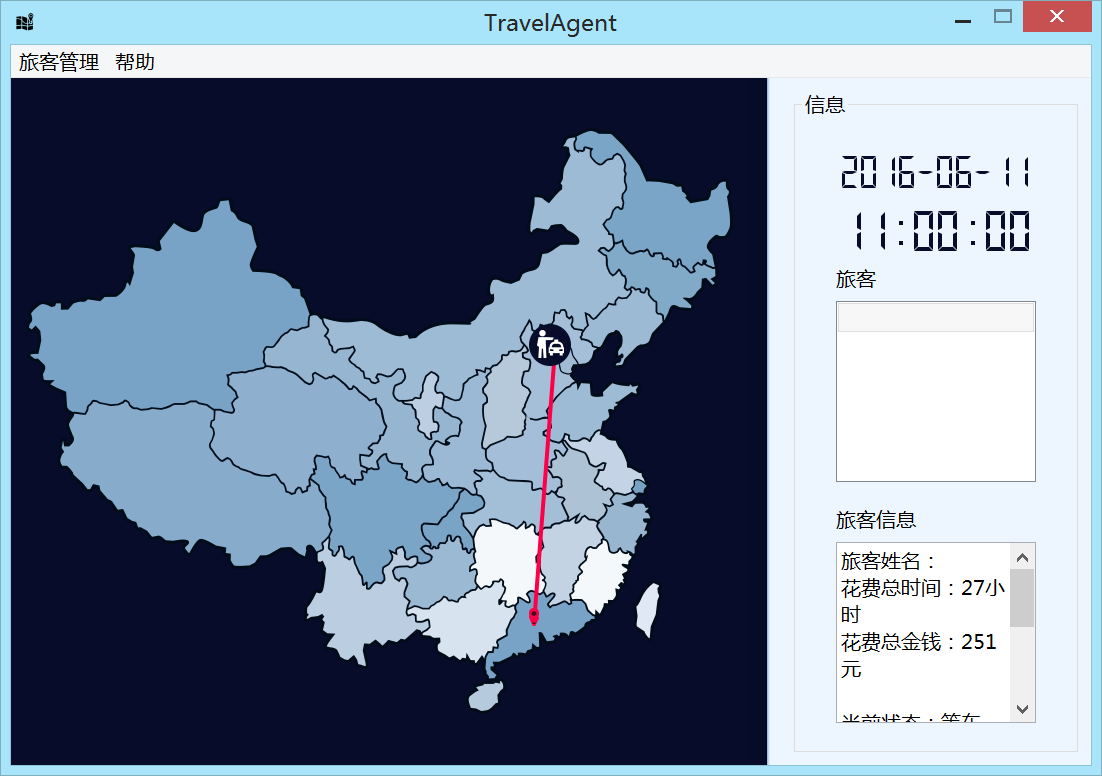
**结果正确**

* 1. 测试样例9：



**测试点**：旅客姓名为空格

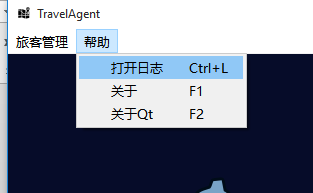
**测试结果**：



**运行正常**

* 1. 测试样例10：

ctrl + L或点此按钮：



**测试点**：查看日志功能

**测试结果**：



**结果正确**

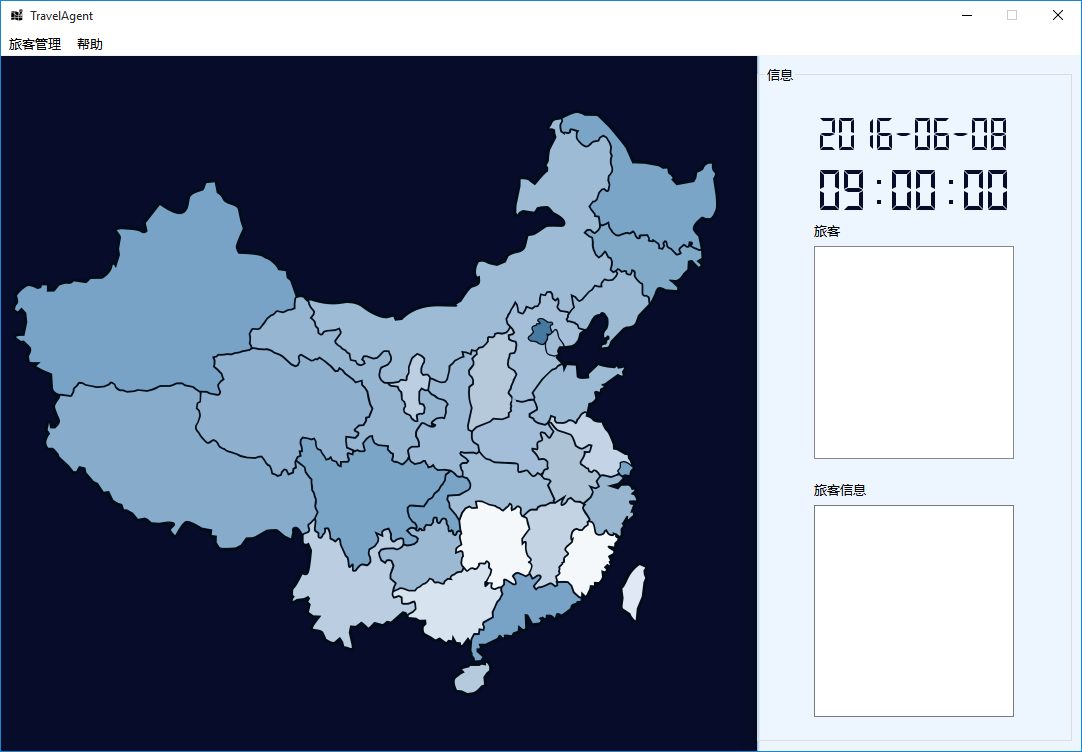
1. 评价和改进意见

该系统具有有精美友好的图形化界面，完备的日志记录及良好的健壮性，支持每个旅客对途径多个城市及最短时间、最小费用、限时最小费用旅行策略的需求，并支持旅客中途更改需求，能在零至数十毫秒内正确高效地对旅客需求制定出相应方案。

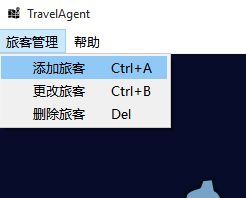
该系统还可在用户对交通路线图的定制化上作进一步完善。

1. 用户使用说明

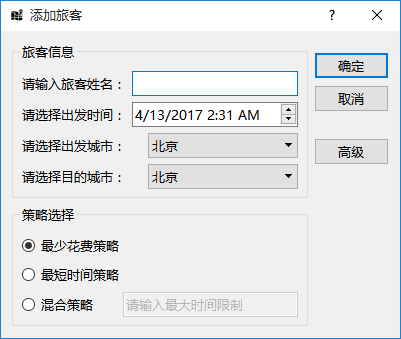
欢迎使用该旅行查询与模拟系统，系统用户界面如下：



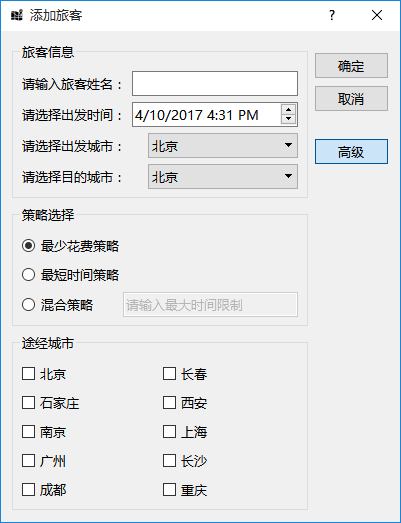
要添加旅客，按ctrl + A或点击此按钮：



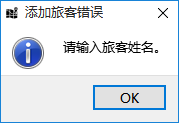
您将看到添加旅客对话框，可直接填写各项基本旅行需求：

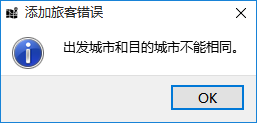


若有必须途径的城市需求，请点击【高级】按钮，您将看到拓展的添加旅客对话框，可在上点选您想途径的城市：

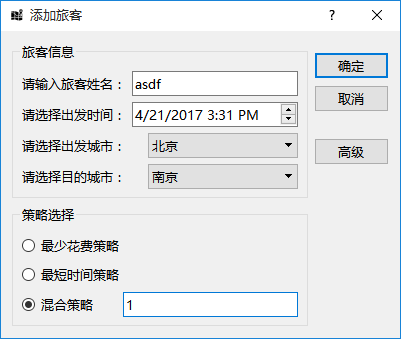


若你信息填写不完整或选择了相同的出发地与目的地，您将收到错误提示，请正确地填写：

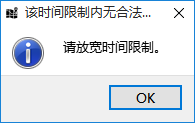




若您想采用第三种策略，时限过小有可能找不到合法路径，如下例：



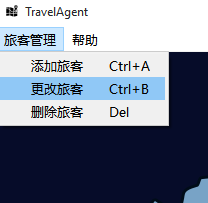
您将收到该提示，请放宽您的时间限制：



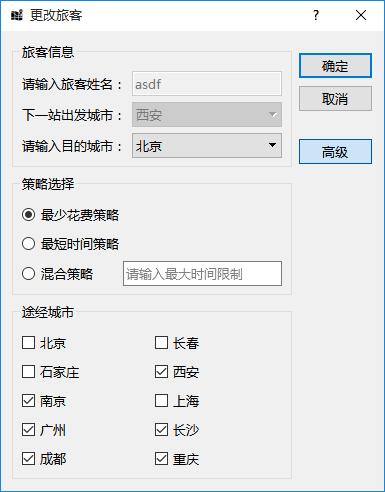
添加旅客成功后，您将看到为该旅客生成的旅游路线。在系统时间达到旅客的出发时间时，系统将开始模拟该旅客的旅行，系统右侧的信息框图将实时地显示该旅客的具体状态，如姓名、旅行总时间、总费用、当前所在航班或车次、到站时间、下一车次等等：



在旅客旅行途中，若要更改某个旅客的需求，您可按ctrl + B或点击此按钮：



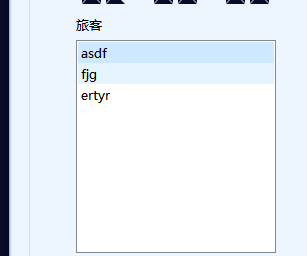
你将看到此对话框，请填写您的新旅行需求：



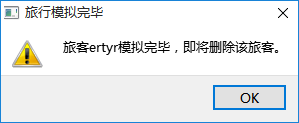
填写成功并确定后，在旅客到达该班航班或车次的目的地时，其旅行方案将发生更改：



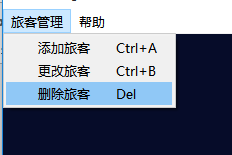
你可添加多名旅客到该系统中，请在右侧旅客列表框中选择您想查看状态模拟的旅客：



某位旅客模拟完毕后，将被从系统中删除：



您也可按Del或点击此按钮将某位旅客从系统中删除：



若要查看日志，您可按ctrl + L或点击此按钮：



系统将弹出自开始运行至当前时刻所有状态变化的日志文件供您查看：



关闭日志文件后系统将继续正常运行。

该系统主要功能演示完毕，祝您使用愉快，拥有一段好旅程：）

