**“明旭舰航”项目说明书**

# 一、设计任务描述：

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某指定旅客所处的地点和状态（停留城市/所在交通工具）。

# 二、功能需求说明及分析：

**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

有沿途到站及票价信息

两站之间不能总是一班车次相连

3.旅客需求涵盖：起点、终点、途经某些城市和旅行策略

4.旅行策略包括：

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：在规定时间内费用最少

5.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)，系统时间精确到小时

6.不考虑城市内换乘交通工具所需时间

7.建立日志文件，对旅客状态变化和键入等信息进行记录

8.用图形绘制地图，并在地图上反映出旅客的旅行过程。

**需求分析：**

1.城市选取：为了更好地体现出算法的正确性和高效性，我们选取了距离较近的15所城市：北京、重庆、长沙、合肥、杭州、济南、南京、上海、石家庄、苏州、天津、武汉、西安、徐州、郑州，使得两所城市之间有足够多的车次/航班。

2. 汽车、火车和飞机的时刻表建立：基于相邻城市之间才有汽车直达，我们选取了170条车次，其中天津和北京之间的车次最多，所处同一个省份的南京、苏州、徐州间的车次较多；火车时刻表只选取了高铁/动车车次，考虑目前全国的高铁建设已十分发达，几乎所有城市之间均存在直达车次，共选取了火车2981条记录；飞机时刻表除重庆、徐州、苏州外的其他城市均存在较多的航班数，这三座城市的航班数较少，共选取了600条航班记录。

3.旅客需求及旅行策略：

最少费用策略：旅客选取出发地和目的地，通过程序计算给出总费用最少的出行策略，无需考虑时间，可从整个时间表中寻求策略。

最短时间策略：旅客选取出发地和目的地，从当前时间开始计算，查找用时最短的出行策略，无需考虑费用，优先从当前时刻的时间表寻求策略。

规定时间内花费最短：旅客选取出发地、目的地及规定时间，在规定时间的时刻表内，查询花费最少的策略。

PS:在计算时间时，不考虑城市内换成交通工具的时间，如在某一城市中转，刚到达此城市时有另一班车可以乘坐，则可以乘坐，不考虑换乘时间。

4.时间控制：系统时间精确到小时，每10秒向前推近一个小时，当有查询操作或新增旅客时，可以暂停时间轴的推进。

5.建立日志文件：对旅客的信息进行存储，包括出发地和目的地，出行策略，并可以随时查询。

# 三、总体方案设计说明：

**软件开发环境：**

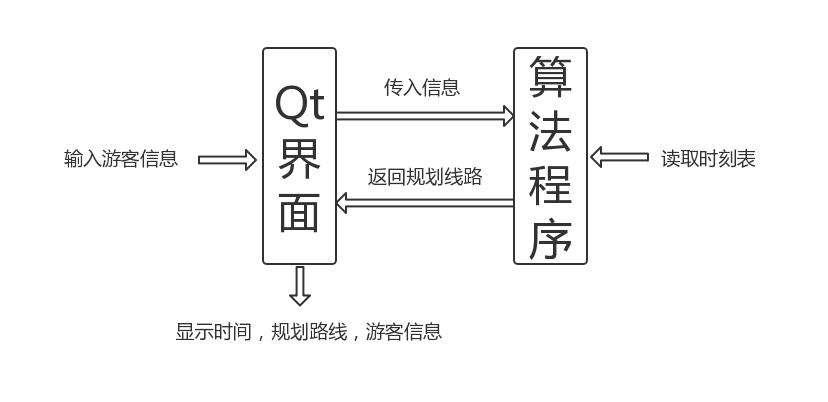
编程语言：C++

代码开发环境：Visual Studio 2017

可视化实现：Qt 5.0

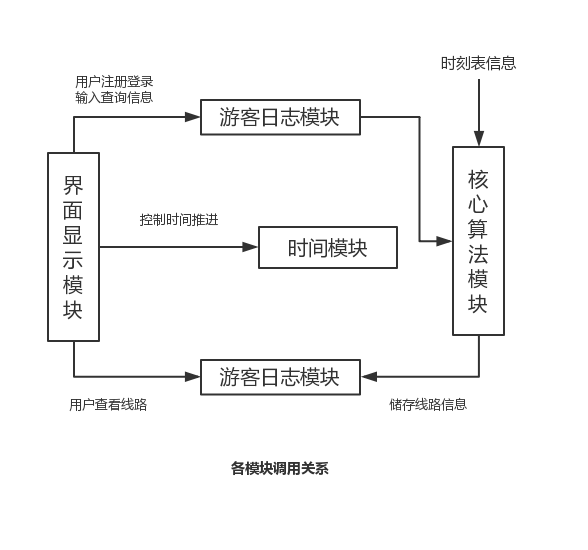
数据库：MySql 8.0.16

**总体结构：**



在Qt可视化面板中，时间按时间轴向前推进，此时用户可输入旅客信息进行线路规划，在接收到信息后传入后台算法程序中进行计算，得出最佳路线后返回给可视化界面进行显示，在程序运行过程中，可控制时间轴的推进及暂停，以及查询某位旅客的信息及当前状态。

**模块划分：**



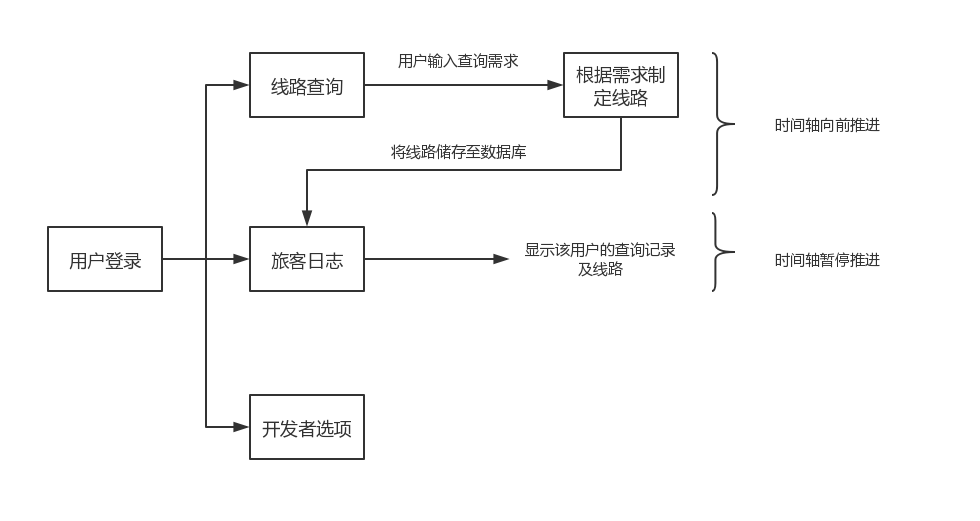
界面显示模块：面对用户的可视化界面，读取用户信息和查询需求，显示相关内容。

时间模块：控制时间推进，每10s向前推进一小时，旅客查看线路时暂停推进。

核心算法模块：读取时刻表，接收并存储游客信息，根据要求规划相应线路并进行保存。

旅客日志模块：用来存储旅客个人信息以及查询线路信息，保存至数据库中，每个游客只能查看本人的线路信息。

**流程图：**



# 四、数据结构及数据字典：

**数据结构：**

**Vihicle类，用于储存时刻表信息：**

class vehicle {

public:

QString Vec[1000]; //交通工具名称：car/train/plane

QString Departure\_Station[1000]; //储存出发地

QString Terminus[1000]; //储存目的地

QString vehicle\_number[1000]; //储存车次或航班号

QString time\_begin[1000]; //储存出发时间

static int reque[maxnum]; //

double time\_cost[1000]; //储存到达时间

double money\_cost[1000]; //存储花费金额

friend class city; //city类的友类

friend class Passenger; //passenger类的友类

};

**route类，用于储存规划线路：**

class route {

public:

QString Vec; //储存交通工具的种类

QSring Departure\_Station; //储存起始地

QString Terminus; //储存目的地

QString vehicle\_number; //储存车次/航班号

QString time\_begin; //储存出发时间

double time\_cost; //储存花费时间

double money\_cost; //储存花费金额

friend class city; //city类的友类

friend class Passenger; //Passenger的友类

};

**Passenger类，储存游客信息：**

class Passenger {

public:

QString name; //储存乘客姓名

QString current\_city; //储存乘客当前所处城市

QString Departure\_place; //储存乘客的起始地

QString Terminus; //储存乘客的目的地

route Stortest\_time; //储存用时最短的路线

route Min\_cost; //储存花费最少的路线

route time\_min\_cost; //储存规定时间内最少花费的路线

vehicle current\_vheicle; //储存乘客当前乘坐的交通工具

};

**City类，存储城市信息，包括三种交通工具的时刻表及三种规划线路：**

class city {

public:

string City\_Name; //城市名称

vehicle train; //火车时刻表

vehicle car; //汽车时刻表

vehicle plane; //航班时刻表

route min\_cost[15]; //最少花费线路

route min\_time[15]; //最少时间线路

route time\_and\_cost[15]; //规定时间最少花费线路

void init\_city(string file\_1, string file\_2, string file\_3, string city\_name); //建立某个城市的时刻表信息，file\_1,file\_2,file\_3分别表示火车、汽车、飞机的数据文件名，city\_name表示出发城市的名字

};

**城市坐标结构图，储存城市在地图上的坐标**

typedef struct CityCoordinate{

int x; //城市的X坐标

int y; //城市的Y坐标

}CityCoordinate;

**花费时间类，根据出发和到达时间存储花费时间：**

class Time\_cost {

public:

double begin\_time; //开始时间

double time\_cost; //花费时间

};

**数据字典：**

#define maxnum 16 //最大城市数

#define maxint 9999.0 //当两地不直达时置为9999

string ADDRESS[15] = { "北京", "重庆", "长沙", "合肥", "杭州", "济南", "南京", "上海", "石家庄", "苏州", "天津", "武汉", "西安", "徐州", "郑州"}; //用来储存城市名称

string address[15] = { "BJ", "CQ", "CS", "HF", "HZ", "JN", "NJ", "SH", "SJ", "SZ", "TJ", "WH", "XA", "XZ", "ZZ" }; //用来对应城市名称

# 五、各模块设计说明：

**1.界面显示模块：**

该模块主要实现Qt界面可视化，通过监听各事件判断该执行的操作，是其他模块实现的基础，通过点击及输入向后台传入数据，从而达成前后端的交互，通过该模块实现对日志模块和核心算法模块的调用，以及时间模块的使用，主要包括mainwindow.cpp、paintlabel.cpp两个程序：

**mainwindow.cpp：主界面显示及各种点击事件的处理，实现可视化。**

相关函数：

MainWindow::MainWindow(QWidget \*parent)

函数功能：实现可视化功能，显示界面。

void MainWindow::timeUpDate()

函数功能：显示当前运行时间。

**paintlabel.cpp：根据得到的路线在地图上进行绘制。**

相关函数：

void PaintLabel::paintEvent(QPaintEvent \*)

函数功能：根据坐标绘制线路，便于旅客查看结果。

**2.时间模块：**

该模块较为简单，实现时间的推进，在查询线路页面及主页面按10s每小时的速度推进，而在查看旅客日志的时候，暂停时间推进，主要通过time\_copy.cpp实现：

time\_copy.cpp：显示当前时间，每10s向前推进一小时。

相关函数：

void time::doLongTimeWork()

函数功能：实现时间每10s向前推进一小时，并传给显示界面。

**3.核心算法模块：**

该模块为程序的核心模块，实现了三种情况下的最佳线路规划，从界面显示模块获取旅客的需求，进行路线规划，并将线路存储至数据库（旅客日志模块）便于用户查看，以下为三种算法的主要思想及特点：

**最少花费方案：**首先根据每个城市的时刻表中的费用信息，建立15×15的最小花费矩阵，每个点代表从城市i出发到城市j的最少花费，当两个城市不可达时，花费设为maxint（9999），接着调用Dijkstra算法求得起始城市到目的城市的最少花费路径，并将经过节点的详细信息进行存储，包括出发时间、到达时间、花费、所乘坐交通工具，按照时间推移在界面中显示旅行过程。

**特点：**实现简单有效，构建了最小花费矩阵可直接使用Dijkstra进行求解，而且便于存储途经节点信息。

**最短时间方案：**根据每个城市时刻表中的时间信息，建立15×15的最短时间矩阵。时间牵扯是否跨天的问题，由于假设每天的时刻表都相同，故在求取最少花费时间时会判断当前车次结束时间与下一车次开始时间前后是否跨天进行加24来求取。每个点代表从城市i出发到城市j的最少时间，当两个城市不可达时，花费设为maxint（9999）。然后根据得到的最短时间矩阵调用Dijkstra算法求得起始城市到目的城市的最短时间路径，并将经过节点的详细信息进行存储，包括出发时间、到达时间、花费、所乘坐交通工具，按照时间推移在界面中显示旅行过程。

**特点：**与算法一类似，建立了最短时间矩阵，在构建时考虑跨天的问题，保证了结果的正确性，调用Dijkstra算法进行求解，简单有效，便于存储途径节点信息。

**规定时间最少花费方案：此算法的前一部分与算法一相同：**首先根据每个城市的时刻表中的费用信息，建立15×15的最小花费矩阵，每个点代表从城市i出发到城市j的最少花费，当两个城市不可达时，花费设为maxint（9999），接着调用Dijkstra算法求得起始城市到目的城市的最少花费路径。接着计算所得路径的花费时间，判断是否在规定的时间内，若在规定时间内：则该路径即为规定时间最少花费路线；若不在规定时间内：则应该为线路中的某一段或者某几段影响了花费时间，先将路线的第一段连接的两城市间花费置为不可达态，递归调用算法三，得到一条路径后进行判断是否为符合条件的最优解，若符合则返回该线路，否则将路线第二段连接的两城市间花费置为不可达态，此时恢复第一次（之前）设置的不可达态为实际花费，接着递归调用该算法，对之后得到每一段进行相同的处理，直到得到符合条件的最优解或判断得到的时间为不可能值时返回，将经过节点的详细信息进行存储，包括出发时间、到达时间、花费、所乘坐交通工具，按照时间推移在界面中显示旅行过程。

**特点：**该算法对问题进行了有效的拆分，首先得到一条最少花费路径，接着根据时间判断是否符合条件，当不符合条件时对路径的每一段单独修改，逐步求得最优解，在存在最优解的情况下必能找到，若不存在最优解则通过时间的可能性判断算法是否可以结束，效率较高，且实现起来较为简单。

**enquiry.cpp：包括三种方案的核心算法，根据需求得到最佳路线。**

相关函数：

void city::init\_city(QString file\_1, QString file\_2, QString file\_3, QString city\_name)

函数功能：读取汽车、火车、飞机的时刻表，初始化某一城市的时刻表信息。

void Dijkstra(int n, int v, double \*dist, int \*prev, double c[maxnum - 1][maxnum - 1])

函数功能：Dijkstra算法的实现，n表示矩阵的行列数，v表示起始点，dist[i]表示起始点到i的距离，prev[i]表示i的前一个节点号为prev[i]，c[ ][ ]代表矩阵

double searchpath\_3(int v, int u, int hour, int minute)

函数功能：根据当前时间查询线路花费的时间，v代表起始点，u表示终点，hour、minute表示开始查询的小时、分钟

void searchPath\_2(int \*prev, int v, int u, int day, int hour, int minute, double t)

函数功能：根据当前时间和规定时间查找两城市间的路径，v代表起始点，u表示终点，day、hour、minute表示开始查询的天数、小时、分钟，t表示规定时间

void find\_limit\_time\_min\_cost(int departure, int terminus, int hour, int minute, double limit, double cost[maxnum - 1][maxnum - 1])

函数功能：查询规定时间内最小花费线路，departure表示起始站，terminus表示终点站，hour、minute表示查询的小时和分钟，limit表示用户的规定时间，cost[ ][ ]表示最小花费矩阵

double find\_min\_cost\_cities(int A, QString B, int j, int hour, int minute)

函数功能：查询最小花费线路，A是出发地城市,B是目的地城市名 begin\_time 表示当前旅客的查询时间，返回花费金额。

void Dijkstra\_2(int n, int v, double \*dist, int \*prev, Time\_cost c[maxnum - 1][maxnum - 1], double start\_time)

函数功能：有时间限制的Dijkstra算法的实现，n表示矩阵的行列数，v表示起始点，dist[i]表示起始点到i的距离，prev[i]表示i的前一个节点号为prev[i]，c[ ][ ]代表矩阵，start\_time代表开始时间。

int find\_address(QString CITY)

函数功能：根据下标查找对应城市。

Time\_cost find\_shortest\_time\_cities(int A, QString B, int j, int hour, int minute)

函数功能：查找最少花费线路，A是出发地城市,B是目的地城市名 begin\_time 表示当前旅客的查询时间，返回花费时间。

void enquiry::plan\_1(QString start\_city, QString end\_city, int day, int hour, int minute)

函数功能：方案一（最少花费策略）的实现，start\_city表示起始站，end\_city表示终点站，day、hour、minute表示查询的天数、小时和分钟

void enquiry::plan\_2(QString start\_city, QString end\_city, int day, int hour, int minute)

函数功能：方案二（最少时间策略）的实现，start\_city表示起始站，end\_city表示终点站，day、hour、minute表示查询的天数、小时和分钟

int enquiry::plan3(QString start\_city, QString end\_city, int day, int hour, int minute, double limit)

函数功能：方案三（限定时间最小花费策略）的实现，start\_city表示起始站，end\_city表示终点站，day、hour、minute表示查询的天数、小时和分钟，limit表示用户的规定时间

**4.旅客日志模块：**

该模块负责记录旅客信息以及查询得到的最佳线路，包括用户的注册登录信息存储，通过三种算法得到的最佳线路，每个用户只能查询自己查询得到的线路，主要包括login.cpp、regist.cpp、record.cpp三个子程序：

**login.cpp：实现用户的登录**

相关函数：

login::login(QWidget \*parent)

函数功能：连接数据库进行查询。

int login::vlogin(QString username,QString password)

函数功能：根据输入的用户名和密码对查询是否存在该用户，存在且匹配返回1，密码错误返回2，用户名不存在返回3。

**regist.cpp：实现用户的注册**

相关函数：

bool regist::gameRegist(QString username,QString password)

函数功能：根据用户输入的用户名和密码对创建新用户，true为创建成功，false为用户名已存在。

**record.cpp：查询旅客的线路记录**

相关函数：

void record::on\_pushButton\_clicked()

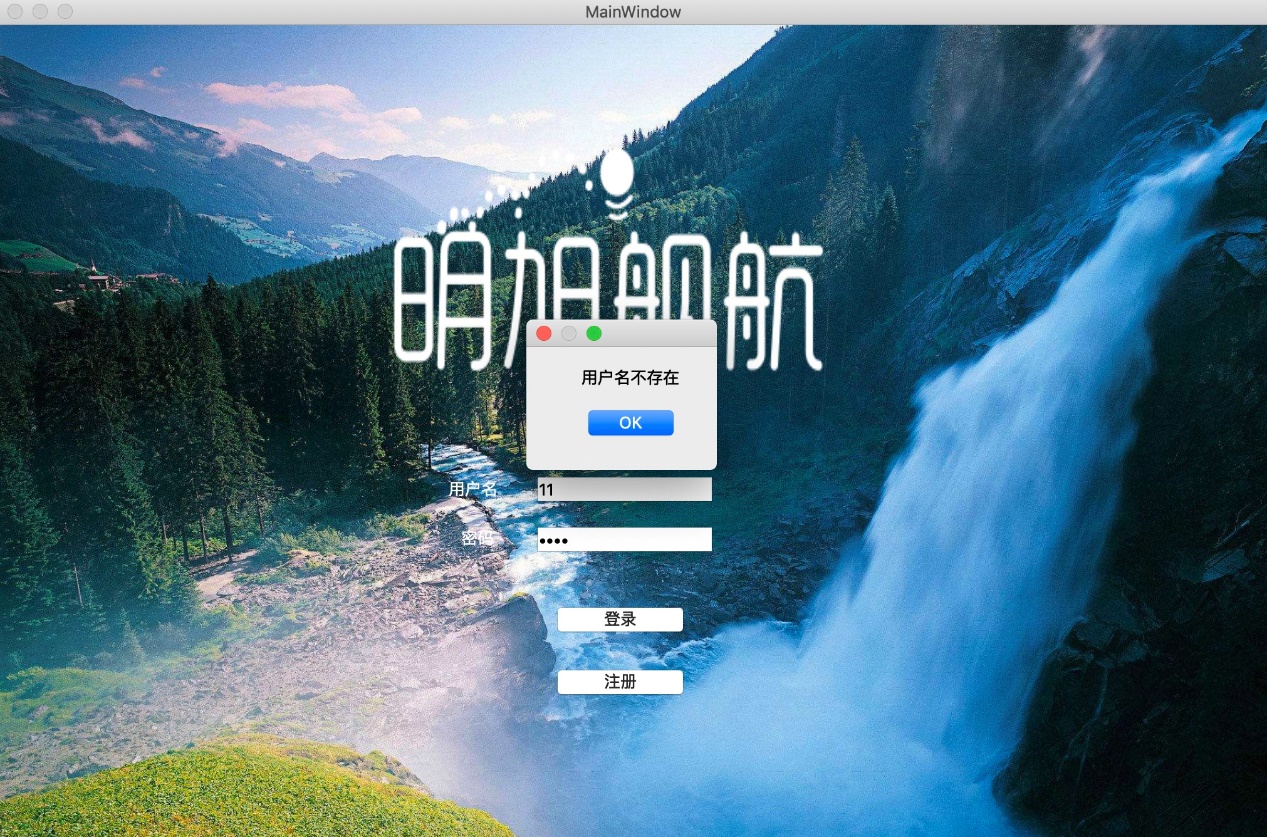
函数功能：判断单击事件发生后，在数据库中查询该用户的线路查询记录并显示。

# 六、范例执行结果及测试情况说明：

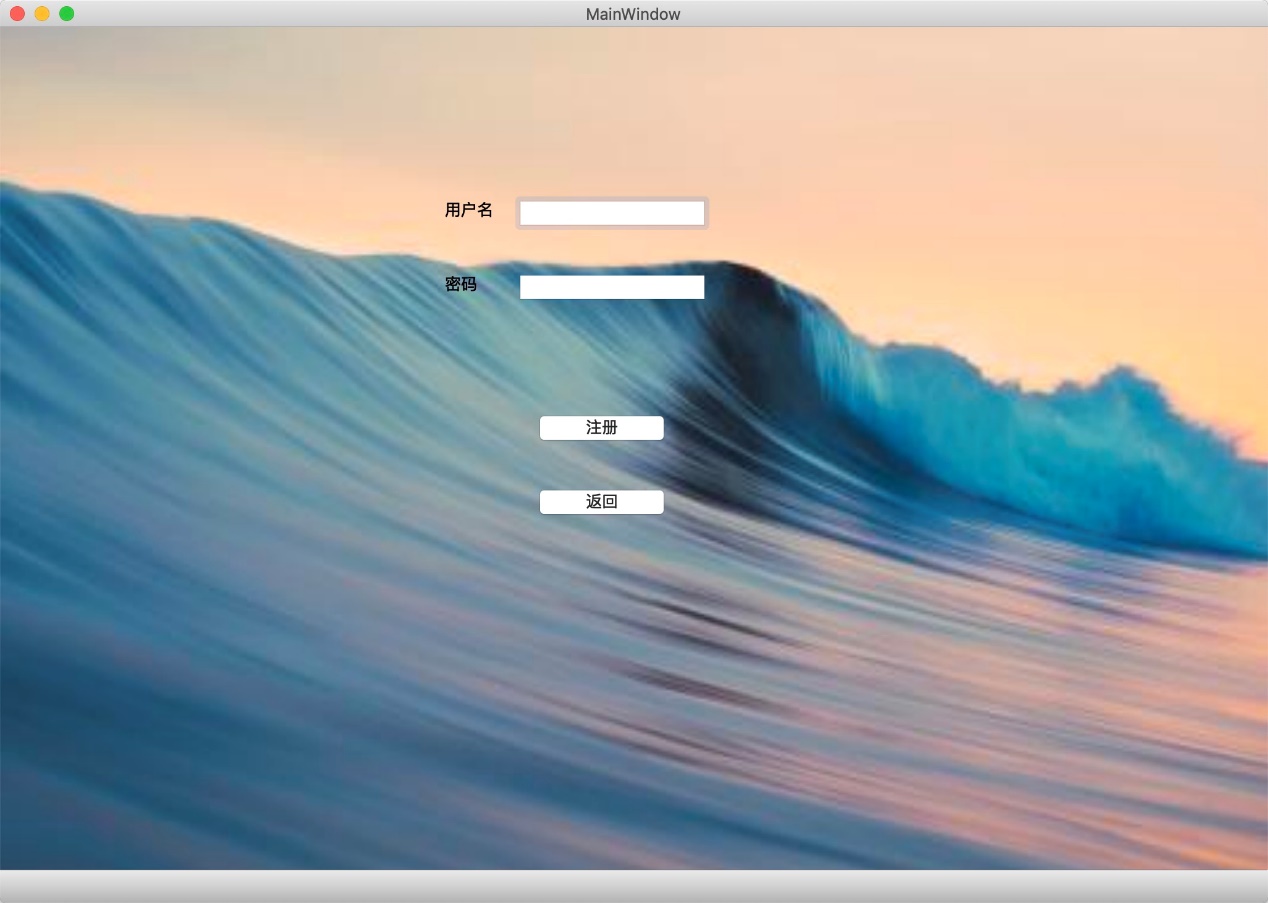
主界面显示：用户登录/注册



测试输入不存在的用户名或用户名与密码不匹配的情况：

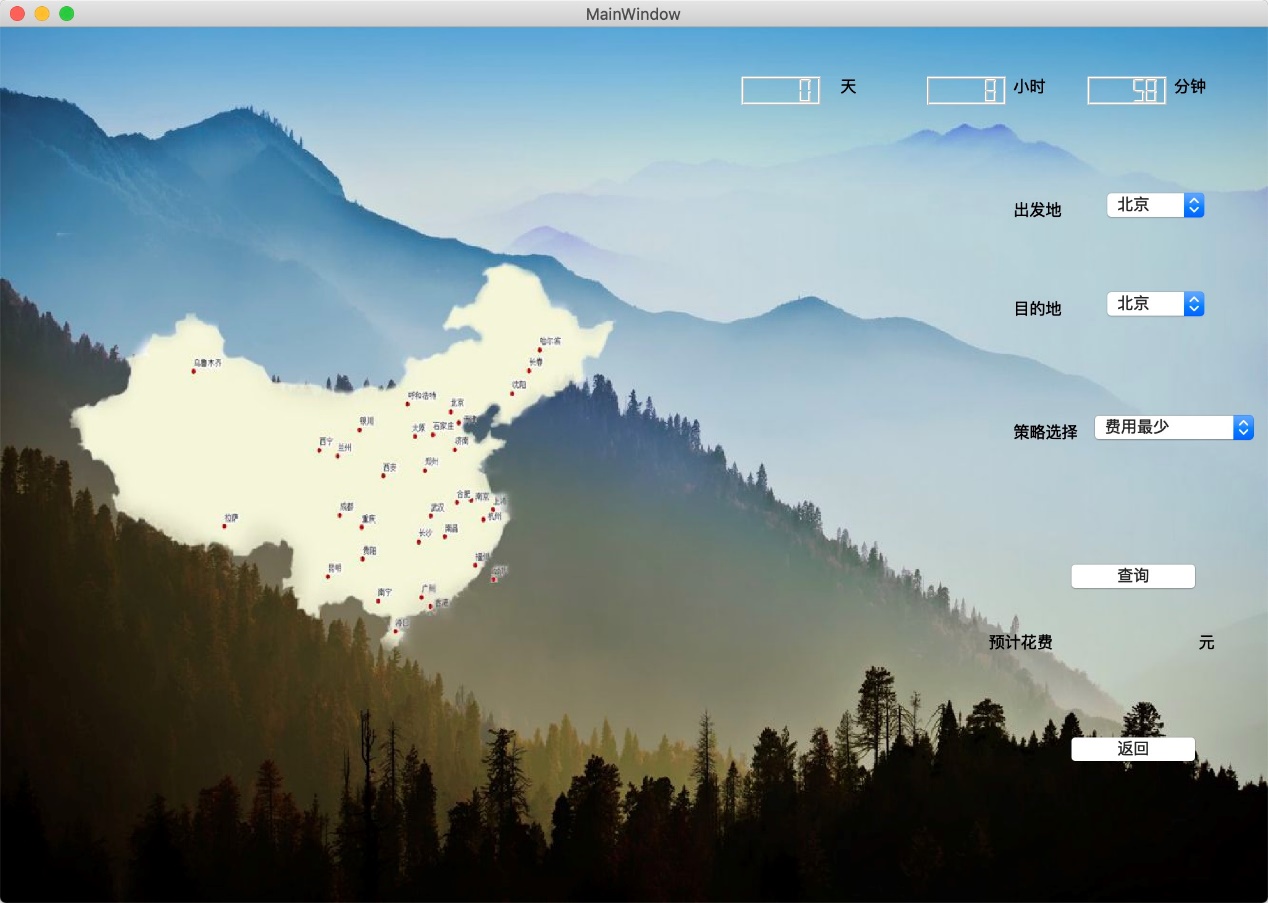


用户注册界面：

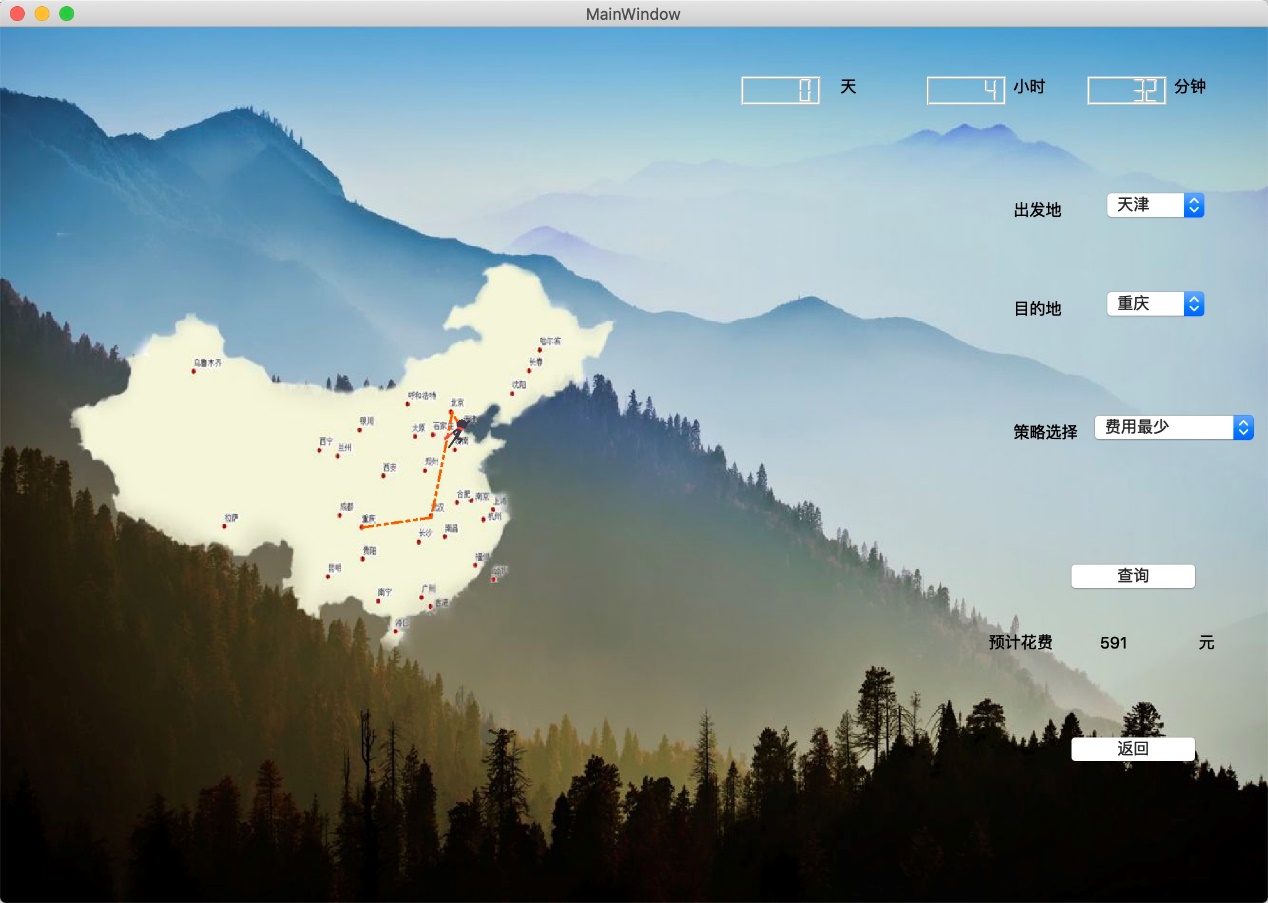
登录成功后显示选项：路线查询，日志查询及开发者选项，右上角显示当前时间：天数—小时—分钟，显示当前用户的用户名：



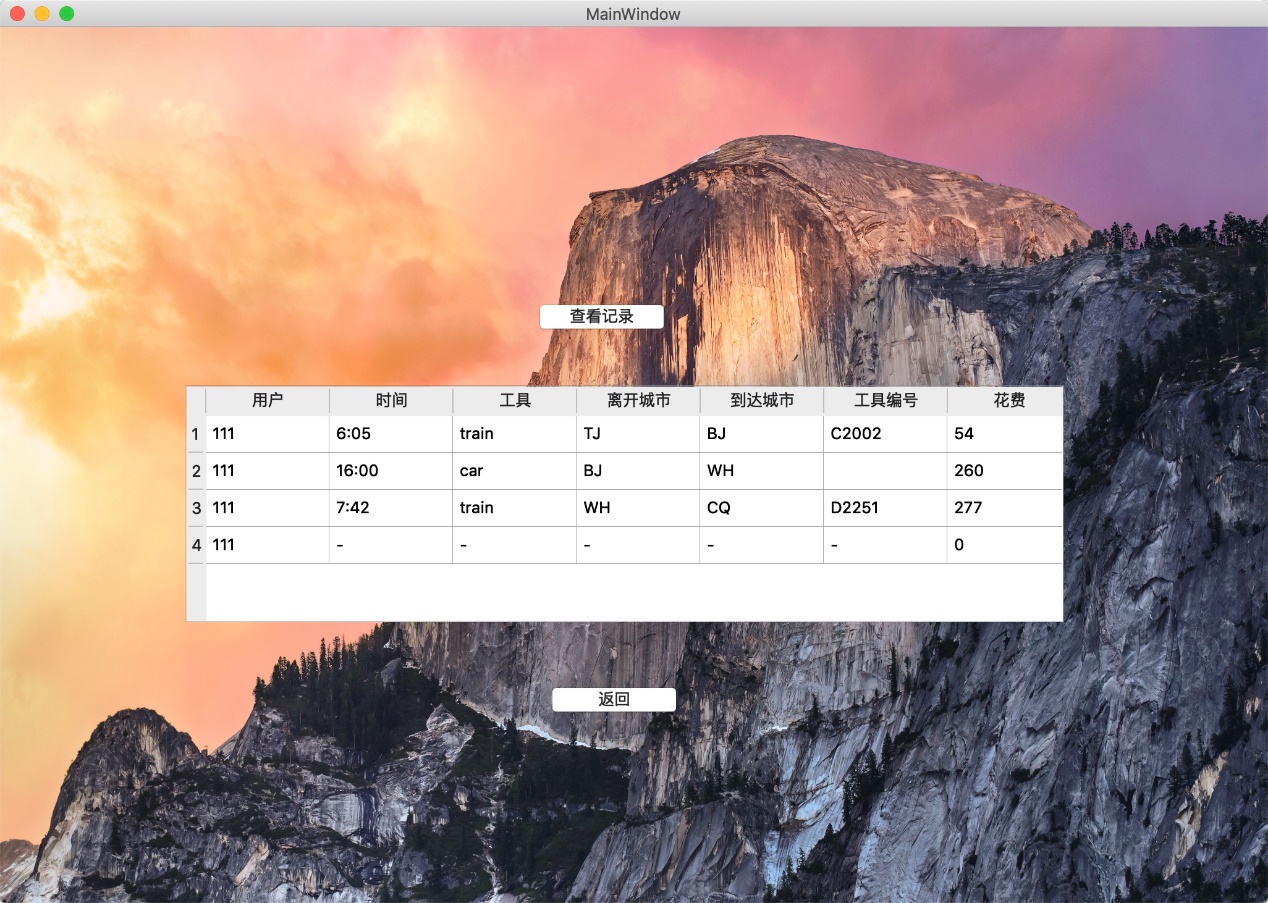
路线查询界面：右上角显示当前时间，左侧显示地图信息，右侧为用户选择项，包括出发地和目的地选择，出行策略选择：



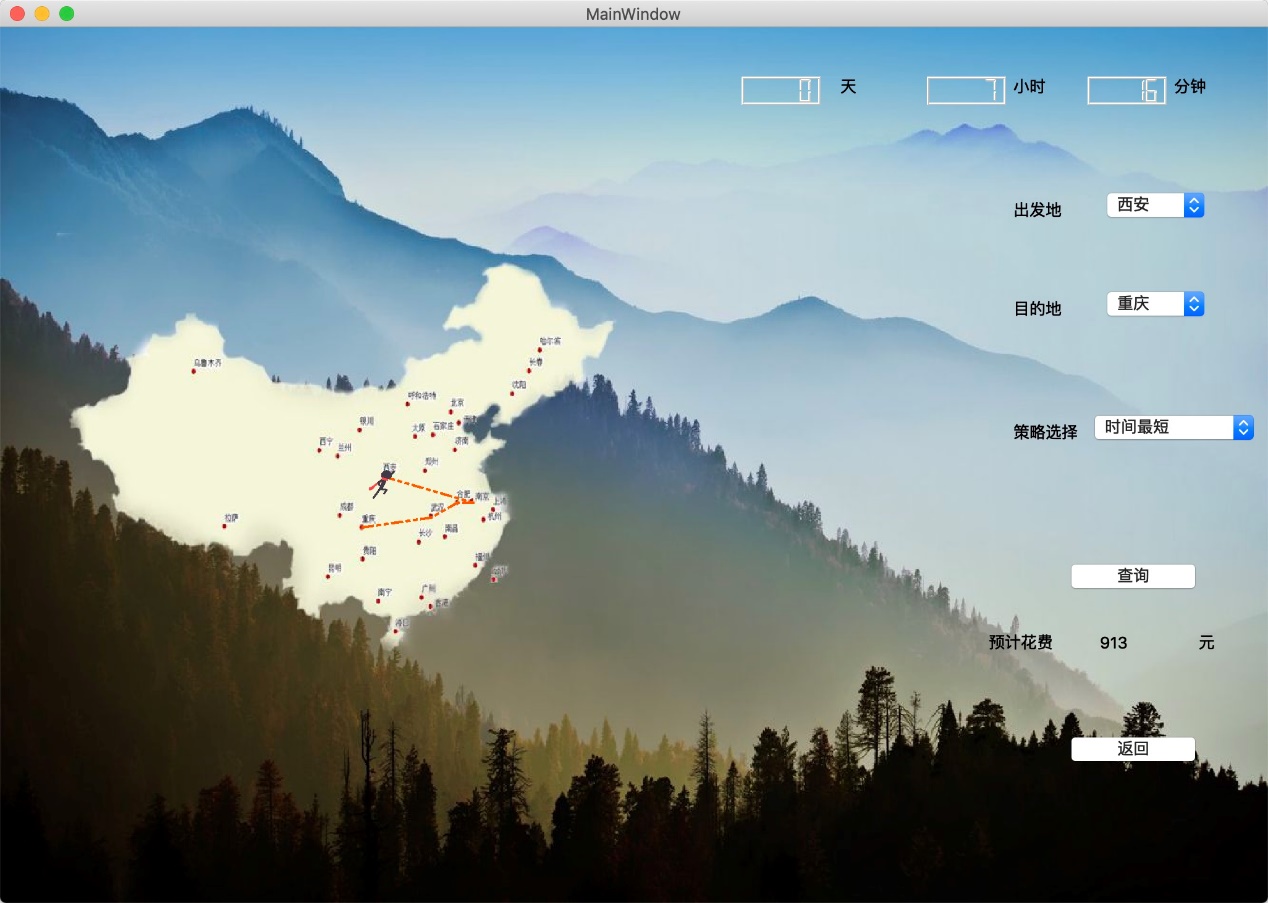
最少费用策略测试，天津——重庆，预计花费591元，此时将在地图上模拟用户的旅行状态，随时间移动：



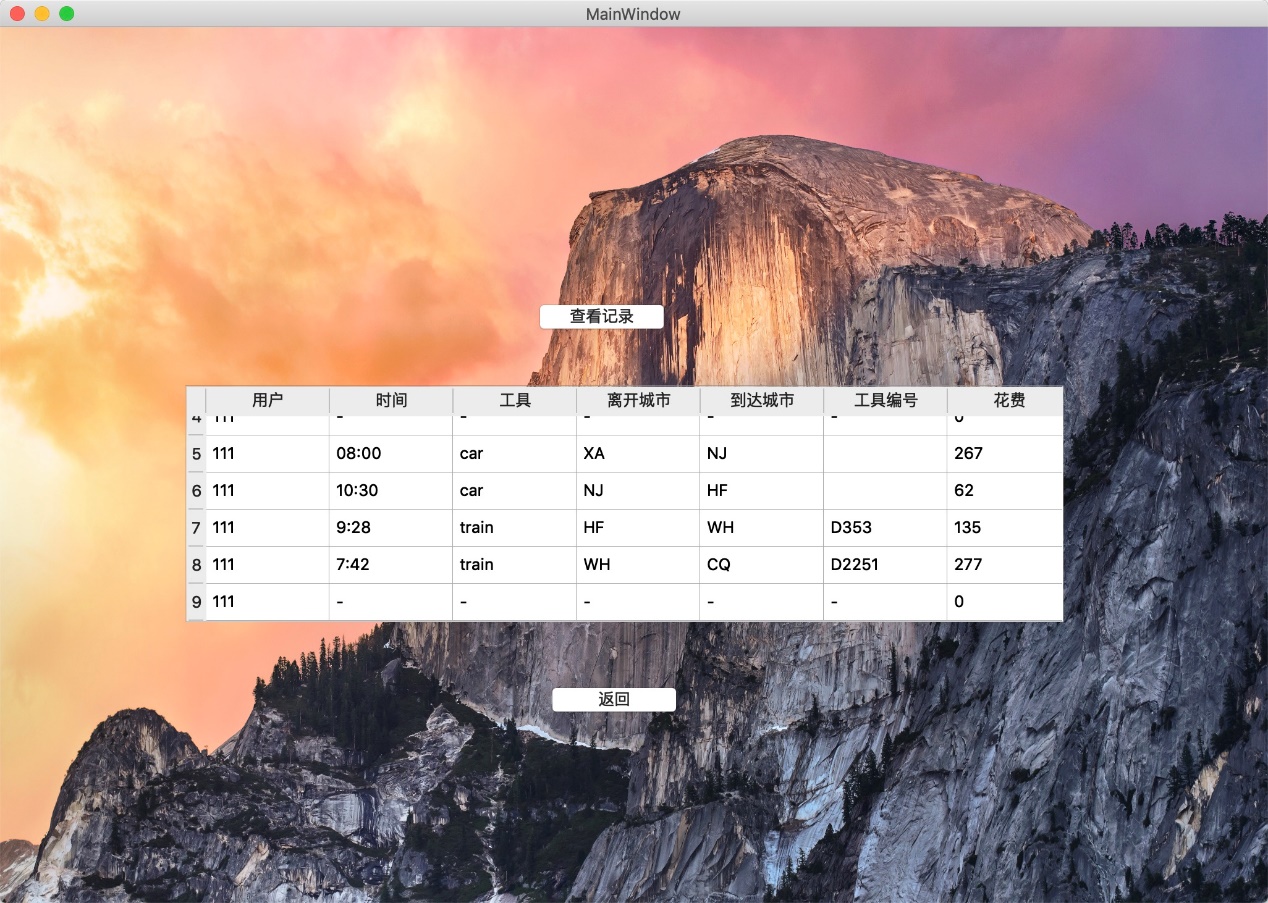
进入日志查询查看线路详细信息：包括查询用户，出发时间，乘坐交通工具，离开城市，到达城市，工具编号及花费：



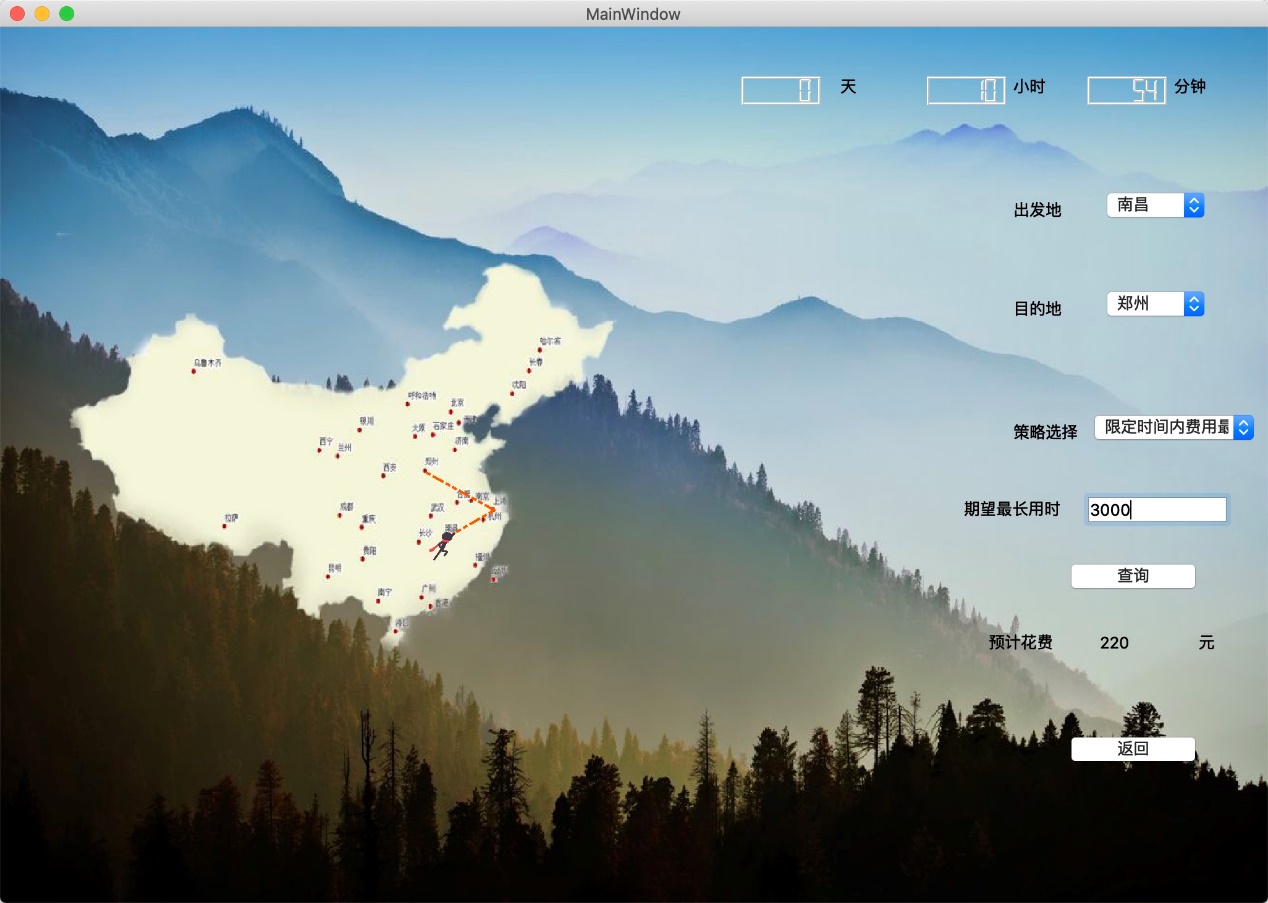
最短时间策略测试，西安—重庆，预计花费913元：



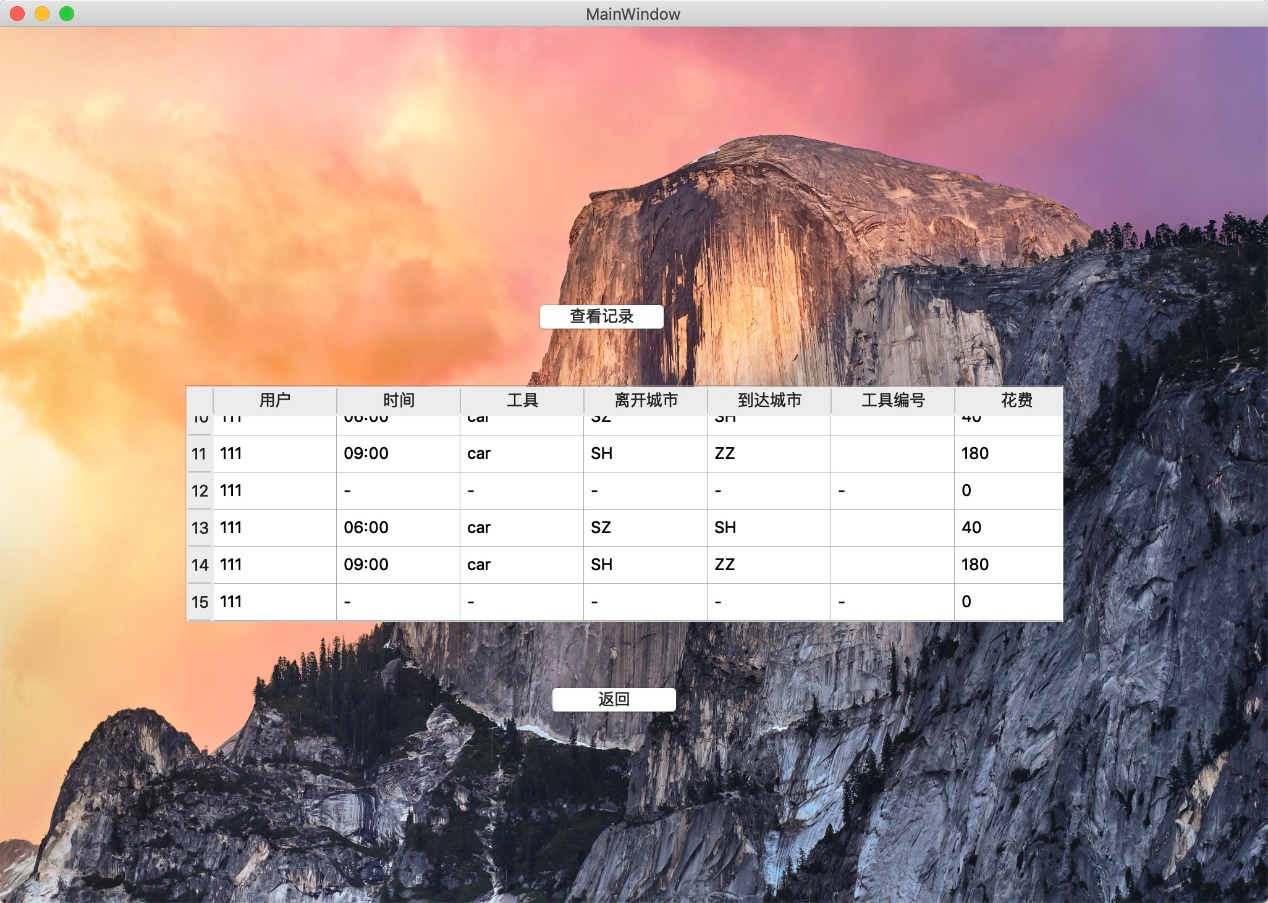
进入日志查询查看线路详细信息：包括查询用户，出发时间，乘坐交通工具，离开城市，到达城市，工具编号及花费：



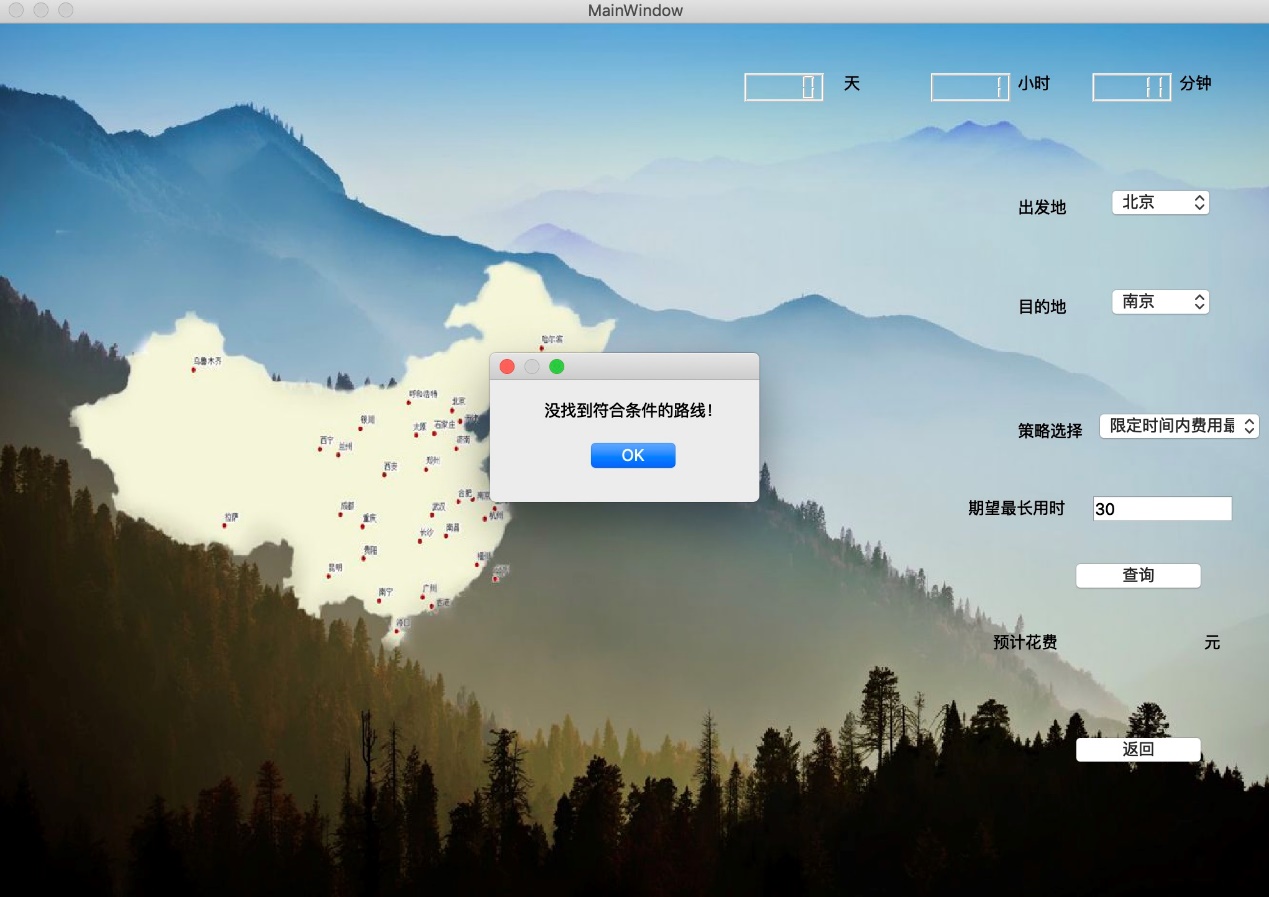
规定时间最少花费策略测试，南昌——郑州，选择第三种策略时，会出现新的输入项：规定时间，需要用户输入，预计花费220元：



进入日志查询查看线路详细信息：包括查询用户，出发时间，乘坐交通工具，离开城市，到达城市，工具编号及花费：



路线不存在情况测试：方案三下输入一个极小的规定时间：



# 七、评价和改进意见：

**项目评价：**

**1.操作简单，条理清晰：**用户只需进行登录（未注册的先注册用户）即可进行后续的操作。日志查询和线路查询两个主板块，日志模块负责记录查询线路得到的详细信息；查询线路模块负责接收用户的查询需求，并据此制定最佳线路，同时在地图上动态模拟旅客的旅行过程，包括路径和当前所处的位置。用户查询界面只允许用户选择而非手动输入（方案三除外），这样可以避免因为输入格式等问题引发的错误，减少了因错误而额外花费的时间。

**2.使用数据库存储用户信息和查询线路，即下次重新登录仍可查询之前的旅行线路：**使用用户登录的目的是为了实现乘客在某次旅行时，可以查看之前的旅行线路详情，这样若有相同的出行计划，则无需再次查询，提高了用户出行的效率；且用户只能查看自己的线路信息，保护了用户的隐私；使用数据库进行数据的存储，便于数据的维护，提高了数据的安全性。

**3.算法效率高，极大减少了因线路查询而耽误的时间：**所涉及到的三个算法均有很高的效率，且保证了计算结果的正确性。算法简单有效，便于我们书写代码的同时，减少了查询时间，做到当前时间和查询时间的同步。

**4.数据集大，城市数目多，更符合实际情况：**软件中涉及15座城市，时刻表包括3000条动车/高铁记录，600条航班记录和170条汽车记录，庞大的数据集使得出行路线更符合实际情况，此外对于相邻城市才存在汽车的出行方式，也提高了结果的合理性。

**改进意见：**

**1.应实现中途修改计划功能：**在实际生活中，旅客在旅行途中应可以随时更改旅行线路来满足相应的需求。我们的软件未实现该功能，用户一旦开始旅行，则不能中途修改计划，这大大降低了用户出行的灵活性，必须提前考虑好一切情况，考虑突发情况是否会影响此次旅行。应在旅客的该次旅行未结束时，添加修改计划功能，只需将当前所处位置作为起点，用户重新选择目的地即可。

**2.应实现多用户同时旅行：**我们所开发的软件不支持多用户同时旅行，即当一位旅客在旅行途中，此时若想进行另一位用户的旅行，则必须等待此用户旅行结束或者停止动态模拟该用户的旅行状态，然后切换用户进行另一位用户的线路查询及动态模拟。这种情况下减缓了程序的运行效率，所有用户只能串行查询，无法做到并行运行。故此软件只能面向独立用户，而不支持多用户查询，应实现多用户同时动态模拟，做到面向用户群。

**3.应实现不同交通工具显示不同图标：**在动态模拟旅行过程时，单纯显示用户的位置，而为显示旅客的出行方式（乘坐的交通工具），这样会降低用户的体验，无法直观反映其状态，仅能通过日志查询出行方式。应实现不同出行方式对应不同的图标，从而提高用户体验。