北京郵電大學

数据结构课程设计

<校园导览系统>

兰学超 2019211564
李培阳 2019211580
熊瑞东 2018211612

北京邮电大学课程设计报告

<table>
<tr><td>课程设计
名称</td><td colspan="2">数据结构
课程设计</td><td>学 院</td><td>计算机学院</td><td>指导教师</td><td>张海旸</td></tr>
<tr><td>班 级</td><td>班内序号</td><td colspan="2">学 号</td><td>学生姓名</td><td colspan="2">成绩</td></tr>
<tr><td>2019211314</td><td></td><td colspan="2">2019211564</td><td>兰学超</td><td colspan="2"></td></tr>
<tr><td>2019211314</td><td></td><td colspan="2">2019211580</td><td>李培阳</td><td colspan="2"></td></tr>
<tr><td>2019211314</td><td></td><td colspan="2">2018211612</td><td>熊瑞东</td><td colspan="2"></td></tr>
<tr><td>课
程
设
计
内
容</td><td colspan="6">① 基本内容：设计一个校园导览系统，要求校园导览系统提供查询服务和导航功能。查询服务内容包括输出当前时刻同学所处的地点，周围的教学楼、宿舍楼、餐饮、后勤服务、操场等信息；导航功能包括当某位同学于某一时刻请求导航时，校园导览系统将根据该同学的具体要求为其设计一条线路并输出
② 设计方法：本课程设计采用C++语言对校园导览系统进行开发；
③ 团队分工：系统各模块的分析、设计，以及代码实现和报告中的具体算法书写由兰学超实现，两校区地图的构建、各辅助函数实现和文档编写由熊瑞东和李培阳共同完成。
④ 实验成果：本次课程设计最终完成了具有查询功能和导航功能的校园导览系统。</td></tr>
<tr><td>学生
课程设计
报告
（附页）</td><td colspan="6">见附页设计报告以及程序源代码</td></tr>
<tr><td>课
程
设
计
成
绩
评
定</td><td colspan="6">评语:

成绩:

指导教师签名:
年 月 日</td></tr>
</table>

目录

一、任务描述

大学校园充满着忙忙碌碌的学生和老师们，但是有时候用户宝贵的时间会被复杂的道路和愈来愈多的建筑物的阻碍而浪费，为了不让同学们在自己的目的地的寻路过程中花费更多的时间，我们着手开发这样一款校园导览系统。

对于一个导航系统来说，使用起来方便直观，各地点之间更换快捷，规划出来的路线清晰准确，这些才是客户最核心的需求。在生活节奏愈来愈快的现代社会，“即搜即出”是提升用户体验的一个重点所在。

在位置的选取上，用户们在使用时通常会有如下两大类搜索习惯，一是该用户清楚地知道自己所需要上课的教室或者导员所在的办公室，那就可以只将该用户引导到对应的教学楼或者办公楼，如“教学楼 N 楼”、“东配楼”等，又或者对于有运动需求的同学将其引导到对应的体育场馆，如“体育馆”“游泳馆”“篮球场”等，这些我们只需要提供物理位置即可；二是该用户希望通过搜索某类型服务设施来查询周边存在的该类型设施，并从结果中选择自己心仪的那一个，如用户搜索“超市，系统将展示“地下超市”和“小麦铺超市”两处地点，并由该用户自行选择目的地。并且对于部分需要跨校区上课的同学或者需要往返于两个校区之间的教师，我们就需要对其提供相对应的交通方式，例如校车或者地铁等公共交通方式。

正如高德百度有红绿灯较少和转向较少等多种导航逻辑，校园导

览系统也可以根据同学们的日常习惯提供不同的导航方式，对于时间紧急的同学可以选择最短时间策略，而对于不那么急，不想走太多的路的同学可以选择最短路径。如果这个时候刚好需要帮忙送一个论文或者其他重要东西的时候，也可以找出当前位置到最终目的地并通过某点的最短路径。不可否认的是校园内的基础设施建设条件足以支撑部分骑单车同学来通行，这时候就需要将自行车用户和步行用户区分开来，以免发生意外。由于校车的特殊性，对于校车可能经过的路径也需要进行一定的提示，在用户使用时加以提醒，避免意外的发生。

生活在这里，我们总会对这里有着自己的感情，校园内的学长学姐对某些建筑可能会有一些流传很久的通俗的称呼，那对于新入校的同学可能会不明就里，对于这种情况我们也做出了自己的应对，通过问询学长学姐及日常听闻，我们将这些通俗的称呼和它们所指向的建筑物做了映射，即使搜索这些通俗的称呼也是可以导航到相关建筑的。

既然是校园导览系统，那就不置可否地需要一个实时向用户介绍周边建筑物名称及用途的功能，那也就需要我们不断地读取并记录用户的实时位置，并以此为依据展现出周边建筑物的名称用途，能够让用户更好的游览校园。

二、需求分析

对校园导览系统要求提供以下方面的服务：

（1）地点查询，负责查询相关地点信息和搜索周边一定范围内建筑和服务设施；

（2）导航系统，负责进行两点之间不同策略的导航，或途径某点的多点导航；

（3）模拟时钟，负责系统的时间推进，模拟导航的推进；

（4）地图实现，负责实现两个校区的地图；

（5）日志更新，负责记录用户的状态变化和键入信息。

2.1 地点查询

在地点查询方面应填写的用户需求描述如下：

1、地点检索

① 模糊搜索：允许用户输入的搜索信息与实际的地点名称存在一定差异，并能够根据用户输入的搜索信息匹配最接近的地点名称，例如：输入“教学楼”，可以匹配“教学楼 N 楼”“教学楼 S 楼”等。

② 逻辑名与物理名对应：同时允许用户搜索部分建筑的通俗称呼，例如：用户搜索“学一”，可以匹配“学生食堂”等。

2、 查询信息

① 非即时查询：在未进行导航时，可以搜索并查询某一地点的位置及相关信息，并在图形化展示界面上显示出来。

② 即时查询：在进行导航过程中也可以随时暂停并实时查询所处的位置或同时展示其相关信息，并在图形化界面上展示出来。

3、显示一定范围内建筑信息

① 非即时查询：在非导航状态下，可以根据用户查询的某一地点以及用户设置搜索的最大半径来搜索附近的建筑和服务设施，并显示其相关信息。

② 即时查询：在导航过程中可以实时暂停导航功能，基于用户的当前位置和用户设置的最大半径来搜索附近的建筑和服务设施，并显示其相关信息。

2.2 导航系统

在地点管理方面应填写的用户需求描述如下：

1、两点导航

可以根据用户当前需求选择如下不同的导航逻辑：

① 最短路径：根据用户输入的起点和终点，找出两点之间距离最短的路线并在图形化界面上展示出来。

② 最短时间：根据用户输入的起点和终点，检索两点之间可能的路径，并根据距离/实际速度得出各条路径所需要的时间，将各路径时间进行比对找出最短时间的路径，在图形化界面上加以展示。（实际速度=通行效率*理想速率）

③ 含校内交通工具的最短时间：用户输入起点和终点后可以在校园内随地找到可骑行的单车，且只能在可骑行路段行进，此时根据距离/实际速度得出各条路径所需时间，将各时间进行对比即可得出最短时间的路径，并在图形化界面上展示出来。（实际速度=通行效率*理想速率）

2、多点导航

用户在使用过程中可能存在希望途径多点再到达目的地的情况：

途径多点的最短路径：用户输入起点和终点，并输入希望途径的地点，找出从起点出发途径途径点到达目的地的最短路径，并在图形化界面面展示该路径。

3、课表导航

可以根据学生的课表结合当前时间进行导航：导览系统根据当前时间在当前登录用户的课程表文件内进行检索，自动匹配当前时间的下一节要上的课程信息，并自动找出该逻辑位置对应的相关地点开始导航，在图形化界面上进行展示。

2.3 模拟时钟

1、默认时钟比例

① 用户在校园内使用该系统进行导航时，系统内部时钟与外界时间

存在一定比例，例如，系统内行进一秒的路程，相当于在外界实际行进三十秒。

② 另外在校际进行变化时，时钟比例不同于校园内进行导航时，但仍存在着和外界实际时间变化的比例。例如，系统内行进一秒的路程，相当于外界实际行进十分钟。

2、自行设定比例

除了默认设定的时钟比例外，用户可以根据自身当前需求来自行设定一个时钟比例，例如由原来的 1：30 可以更改为 1：60。

3、模拟导航

用户在使用导航功能时，代表着用户的点可以随着系统模拟时间的流动而改变位置，以此来体现模拟导航的过程。

4、随开随停

用户在使用本系统进行导航的过程中，可以随时停止当前系统内的时钟并进行相关的一系列操作。

2.4 地图实现

通过格式化的方式存储地图信息：

① 对地图进行标点并编号，格式化存储每个点的坐标及名称等相关信息；

② 格式化存储每个点周围的邻接信息；

③ 构建逻辑名称与物理名称的映射表。

2.5 日志更新

① 在用户键入信息或者点击某按钮时，记录该用户的当此活动；

② 当用户状态发生改变时，实时记录其状态的变化。

③ 可以根据时间查询用户该时间的状态及活动信息。

三、概要设计

3.1 开发环境

在校园导览系统的设计与开发上，采用 C++语言进行算法部分的编写工作，基于 QT 进行图形化界面的开发，并在搭载 Windows10 和 IOS 的计算机上进行测试工作。

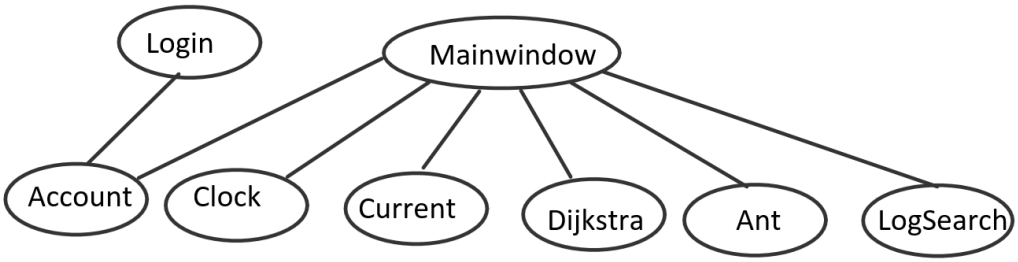
本次实验的编写环境可分为硬件和软件环境，它们分别如下所示：

① 硬件环境：搭载了 Windows10 和 IOS 操作系统的笔记本电脑；

② 软件环境：Xcode　VS code　QT 5.9.9

3.2 总体结构

根据团队对于校园导览系统的需求分析，我们将系统构建为如下结构：



3.3 模块划分

（1）Login 模块：登录模块，提供用户登录窗口，并检测账号密码是否存在于本地文件且是否对应。

（2）Account 模块：账户模块，用于存储用户的账号密码信息与课程时间表信息。

（3）Mainwindow 模块：程序主窗口模块，串联所有核心算法模块，提供用户与系统的交互界面，并以图形化的方式输出。

（4）Clock 模块：时钟模块，提供模拟导航系统的时间。

（5）Current 模块：当前状态模块，提供用户当前状态信息及周边信息。

（6）Dijkstra 模块：迪杰斯特拉算法模块，提供两点间的最短路径的算法，并且结合 BFS 实现搜索一定范围内的所有顶点及其路径的功能

（7）Ant 模块：蚁群算法模块，提供途径多点的最短路径算法。

（8）LogSearch 模块：日志查询模块，提供查询日志文件相关信息的功能。

四、数据结构说明

注：省略了所有的构造函数、析构函数、setter（set 函数）与所有的 getter（get 函数）

4.1 Vertex——顶点

（1）数据结构定义

```
class Vertex
{
private:
    int no;
    qreal x;
    qreal y;
    QString description;
    bool isSpot;
};
```

（2）成员变量意义

int no：顶点的编号；

qreal x：该顶点的 x 坐标；

qreal y：该顶点的 y 坐标；

QString description：该顶点的相关描述信息；

bool isSpot：该点是否为有实际意义的节点的标志。

4.2 Edge——有向边

（1）数据结构定义

```
class Edge
{
```

```
private:
    qreal distance;
    qreal crowdness;
    qreal velocity;
};
```

（2）成员变量意义

qreal distance：该边的总距离

qreal crowdness：该边的拥挤度

qreal velocity：该边可允许通行的最大速度

4.3 current——当前状态

（1）数据结构定义

```
class Current
{
public:
    bool inShahe();
    bool inBenbu();
    bool inBetween();

private:
    qreal x;
    qreal y;

    qreal velocity;
    Priority priority;

    QVector<QMap<int,Edge*>> adj_map;
    QVector<Vertex*> vertices;
    QVector<Vertex*> path;

    Vertex* last_ver;

    int next_ver_index;
```

```
    qreal total_dis;
    qreal current_dis;
    qreal total_pass;
    qreal current_pass;

    qreal total_time;
    qreal total_pass_time;
};
```

（2）成员变量意义

qreal x：当前位置的 x 坐标；

qreal y：当前位置的 y 坐标；

qreal velocity：当前所在边的允许通行速度；

Priority priority：当前的选择策略（优先度）；

QVector<QMap<int,Edge*>> adj_map：邻接表；

QVector<Vertex*> vertices：顶点信息表；

QVector<Vertex*> path：途径数组，包括起点；

Vertex* last_ver：上一顶点，当 current_pass==0 时，last_ver 为当前节点；

int next_ver_index：下一个顶点在途径数组中的下标；

qreal total_dis：路径总距离；

qreal current_dis：当前边的长度；

qreal total_pass：总已走距离；

qreal current_pass：当前已走距离（距离上一个节点的距离）；

qreal total_time：总所需时间；

qreal total_pass_time：总已经历的时间。

（3） 成员函数说明

bool inShahe()：判断当前是否在沙河校区；

bool inBenbu()：判断当前是否在本部校区；

bool inBetween()：判断当前是否在校外。

4.4 Log——日志记录

（1）数据结构定义

```
class Log
{
public:
    void readLog();
    void writeLog();

    void addItem(Clock clk, QString op, QString inf);
    void popItem();

    QVector<QString> search(Clock clk_high, Clock clk_low, con
st QString &ope) const;

private:
    typedef struct ITEM {
        Clock clk;
        QString ope;
        QString info;
    } item;

    QVector<item> log_list;
};
```

（2）成员变量意义

typedef struct ITEM {

Clock clk：日志记录的时间；

QString ope：日志记录的操作信息；

QString info：日志记录的解释性信息；

} item：日志记录的结构体；

QVector<item> log_list：日志记录数组，即日志记录

（3）成员函数说明

void readLog()：读日志文件；

void writeLog()；写日志文件；

void addItem(Clock clk, QString op, QString inf)：

向日志记录数组中写入一条记录；

void popItem()：删除日志记录数组中的最后一条记录；

QVector<QString> search(Clock clk_high,

Clock clk_low, const QString &ope) const：

搜索一定时间范围内的所有日志记录，并格式化为字符串返回。

4.5 Account——账号

（1）数据结构定义

```
class Account
{
public:

    Course gotoclass(Clock clock) const;

private:
    QString id;
    QString password;
    QVector<Course> courses;
};
```

（2）成员变量意义

QString id：账户名；

QString password：密码；

QVector<Course> courses：课程表。

（3）成员函数说明

Course gotoclass(Clock clock) const：

对应当前时间找到正在进行的课程并返回。

4.6 Clock——定时器

（1）数据结构定义

```
class Clock
{
public:
    QString get_clock_str();
    void update_clock();

    int difSec(const Clock &clock);
    bool operator>=(const Clock &clock);
    Clock operator-(int h);

private:
    int sec;
    int min;
    int hour;
    int wday;
};
```

（2）成员变量意义

int sec：秒；

int min：分；

int hour：时；

int wday：星期（0-6，0表示周日）。

（3）成员函数说明

QString get_clock_str()：返回化为格式化字符串的时钟；

void update_clock()：时钟更新，+1s；

int difSec(const Clock &clock)：求两时钟相差的秒数；

bool operator>=(const Clock &clock)：

重载运算符>=：用于判断时间的大于等于关系；

Clock operator-(int h)：

重载运算符-：用于计算时间减去传入的小时数后得到的时间。

4.7 Course——课程

（1）数据结构定义

```
class Course
{
private:
    QString course_name;
    Clock begin_time;
    Clock end_time;
    int classroom;
};
```

（2）成员变量意义

QString course_name：课程名；

Clock begin_time：课程开始时间；

Clock end_time：课程结束时间；

int classroom：教室对应的顶点编号。

4.8 adj_map——邻接表

（1）数据结构定义

```
QVector<QMap<int,Edge*>> adj_map;
```

（2）成员变量意义

QVector 的下标：顶点编号；

QMap 的 key：邻接点的序号；

QMap 的 value：顶点到 Key 的有向边；

4.9 vertices——顶点数组

（1）数据结构定义

```
QVector<Vertex*> vertices;
```

4.10 logic2phy——逻辑名到顶点的映射表

（1）数据结构定义

```
QMap<QString,QVector<int>> logic2phy;
```

（2）成员变量意义

QMap 的 key：逻辑名；

QMap 的 value：对应的物理名的顶点编号数组；

QVector 的值：物理名的顶点编号。

五、详细设计

5.1 Login 模块

登录模块，提供用户登录窗口，并检测账号密码是否存在于本地文件且是否对应。

主要函数及功能如下：

函数原型	功能
`void on_pbt_login_clicked()`	登录按钮对应的槽函数，检查用户输入的账号密码与文件是否对应，若对应则关闭窗口返回 accepted，不对应则弹出提示框。
`bool checkLogin(const QString &id, const QString &pwd)`	参数分别为用户输入的账号和密码，检查账号是否存在于文件中，账号密码是否对应。
`void on_pbt_exit_clicked()`	退出按钮槽函数，关闭窗口，返回 rejected。
`void readAccounts();`	读取文件中的账号密码及课程表信息。

5.2 Account 模块

账户模块，用于存储用户的账号密码信息与课程时间表信息。

主要函数及功能如下：

函数原型	功能
`Course gotoclass(Clock clock)`	参数为当前时间，结合课表信息判断当前是否有课，若有课返回当前在上的课，若没课返回-1。

5.3 Mainwindow 模块

程序主窗口模块，串联所有核心算法模块，提供用户与系统的交互界面，并以图形化的方式输出。

主要函数及功能如下：

函数原型	功能
`void initAction()`	初始化工具栏按钮
`void initToolBar()`	初始化工具栏
`void initTimer()`	初始化定时器
`void readAdj()`	读入地图，存入邻接表
`void readVer()`	读入节点信息表

`void readL2P()`	读入逻辑名到物理名映射表
`void readShuttleSchedule()`	读入班车时刻表
`void currentOutput()`	输出当前状态信息
`bool inShahe(int spot)`	判断传入的点是否在沙河校区
`bool inBenbu(int spot)`	判断传入的点是否在本部校区
`bool inBetween(int spot)`	判断传入的点是否在两校区之间
`void searchNear()`	搜索附近一定范围的建筑并提供行进路线、距离和时间
`void searchSpot()`	搜索单个建筑并提供行进路线、距离和时间
`void findPath()`	两点寻路函数
`void antFind()`	多点寻路函数
`void switchShahe()`	切换至沙河地图
`void switchBetween()`	切换至校外地图
`void switchBenbu()`	切换至本部地图
`void setTime()`	设置时钟加速的结束时间

`void drawPath(const QVector<Vertex*> &path, bool currentIn)`	绘制整条路径，参数分别为要绘制的路径、是否将当前位置绘入路线的标志
`void move()`	更新当前状态并画出行进路线，若跨校区则进行地图切换
`void clear()`	擦除地图上的所有路线
`void setCurrent(const QVector<Vertex*> &path, qreal dis, qreal time, Priority pr)`	设置当前状态，传入的参数分别为导航路径、距离、时间和优先度
`void setCurrentToVer(int no)`	将当前状态移动至指定的顶点
`void updateTime()`	系统时间+1s 并显示在文本框中
`int getFirstShuttleTime(qreal t)`	参数为到达班车上车点所需的秒数，返回到达上车点后还需要的等车秒数
`int getFirstBusTime(qreal t)`	参数为到达班车上车点所需的秒数，返回到达上车点后还需要的等车秒

	数
`void fuzzySearch(const QString &str)`	参数为输入的字符串，执行模糊搜索，将搜索结果记录在数组中，并依此动态创建多个选择按钮
`void mouseDoubleClickEvent (QMouseEvent *e)`	鼠标双击事件，在界面中显示当前点击顶点的相关信息
`void setVelocity(int x)`	设置速度函数
`void clock_slot()`	定时器超时执行的槽函数，包括 currentOutput、updateTime、move 等。
`void cancel()`	清除右侧创建的按钮
`void reenter()`	重新输入函数，清除右侧按钮，清除文本输入框
`void choose_findpath_time()`	选择最短时间路线
`void choose_findpath_dis()`	选择最短距离路线
`void choose_searchnear(int i)`	从搜索附近的所有结果中选择一个作为终点
`void choose_search_fzs(int i)`	从搜索地点文本框的模糊搜索的所有结果中选择一个作为文本

`void choose_fp_begin_fzs(int i)`	从两点寻路起点文本框的模糊搜索的所有结果中选择一个作为文本
`void choose_fp_end_fzs(int i)`	从两点寻路终点文本框的模糊搜索的所有结果中选择一个作为文本
`void choose_ant_fzs(int i)`	从多点寻路文本框的模糊搜索的所有结果中选择一个作为文本
`void antfindNext()`	继续输入多点寻路的下一个途经地点
`void antfindStart()`	多点寻路开始导航
`void antfindStartDis()`	选择多点寻路的最短距离路线
`void antfindStartTime()`	选择多点寻路的最短时间路线
`void stopTime()`	暂停系统时钟
`void logSearch()`	打开日志搜索窗口
`void gotoClass()`	根据当前时间和用户课程表，导航去教室
`void search_fzs(const QString &str)`	对搜索地点文本框中的内容进行模糊搜索

`void findpath_begin_fzs(const QString &str)`	对两点寻路起点文本框中的内容进行模糊搜索
`void findpath_end_fzs(const QString &str)`	对两点寻路终点文本框中的内容进行模糊搜索
`void antfind_fzs(const QString &str)`	对多点寻路文本框中的内容进行模糊搜索

算法分析：

1. 导航线路推进：

用求得的路径数组、距离、时间等信息设置当前状态，表示开始导航。在 clock_slot 中调用的 move 函数中，程序计算经过系统的 1s 后前进的距离，更改当前状态中的 current_pass 及上一顶点、下一顶点等信息。同时将 Qt 的 painter_path 连线到当前状态的坐标。具体推进伪代码如下：

```
Double t = 1;
double v = 当前速度*所在边的拥挤度;
Double d = t*v;
While（当前状态加上 d 足以到达下一顶点） {
    If(下一顶点是当前路径终点) {
        更改当前状态到终点（坐标、已走距离/时间、上一/下一节点、速度=0 等）
    }
```

```
    Else {
        t -= 走到下一节点所需时间;
        更改当前状态到下一顶点（坐标、已走距离/时间、上一/下一节点、当前边距离等);
        v = 所在新边的拥挤度*当前速度;
        d = v * t;
        绘制路线（连线到当前顶点);
    }
}//while
If(没到终点) {
    更改当前状态到下一顶点（坐标、已走距离/时间、上一/下一节点、当前边距离等);
}
绘制路线（连线到当前坐标);
```

2.两点寻路跨校区导航：

由于跨校区校外路段及班车、公交的特殊性，采取分段导航的策略。如果在寻路时发现起止点不在同一校区，即设置跨校区 cross 标志为 1，并进行 3 段寻路（以公交为例）：第一段从起点到当前校区公交上车点；第二段为设定好的公交路线；第三段从目的校区公交上车点到终点。

开始导航后若当前状态到达第一段路线终点，系统自动用第二

段路线设置当前状态，到达第二段终点时同理。

3.途经多点导航：

由于途经多点导航与传统的旅行商问题略有区别：并非遍历所有顶点也并非需要回到起点，故蚁群算法有所改动。

不能以原始邻接表 adj_map 作为蚁群算法的邻接表，会出现两顶点不连通的情况。故应二重循环遍历途经顶点数组，两两之间使用 dijkstra 算法生成有向最短路径，并存于抽象完全图 QMap<int,QMap<int,Dijkstra>> abmap 中（abmap[i][j]存储从顶点 i 到顶点 j 的最短路径、距离、时间）。通过将蚁群算法使用的邻接图设为此 abmap，问题就解决了。但是生成该 abmap 的时间复杂度是 O(n^4)，效率略低，不过总体相比旅行商问题的指数级是十分优秀的，并且准确度较高。

4.途经多点跨校区：

类似的，多点寻路也会面临跨校区的问题。解决方法也是分段寻路，以下以公交为例。

首先将途经顶点数组分为两个：沙河顶点数组和本部顶点数组。然后将沙河公交上车点加入沙河顶点数组（如果原本没有），本部同理。然后对应两个顶点数组分别生成两个 abmap。

随后遍历沙河顶点数组和本部顶点数组，将不同校区的所有途经顶点构建为一个 abmap（沙河顶点 a 到本部顶点 b 的路径由 3

段合成：a 到沙河公交点路径+跨校区路径+本部公交点到 b 的路径）。以该 abmap 进行蚁群算法。

接着将蚁群后得到的路径拆分，由于最短路径的必然性，该路径中必然可分为三段，第一段为沙河校区路径（终点为沙河公交点），第二段为跨校区路径，最后一段为本部路径（起点为本部公交点）。

开始导航后若当前状态到达第一段路线终点，系统自动用第二段路线设置当前状态，到达第二段终点时同理。

5.离开/进入校园自动切换地图的显示：

当前状态校园出口并即将跨校区时，系统自动切换校区是合理的需求。为了易于实现该操作，我们多设置了两个逻辑上的顶点，称之为沙河和本部切地图点，以沙河切地图点为例。

沙河切地图点物理上与沙河校门点重合，即从校门到该点的距离为 0。利用边距离为 0 这一特殊性，我们在 move 函数中加入判断：若当前上一顶点为沙河校门点且当前边距离为 0 且跨校区标志 cross==true，将自动切换地图。本部同理。

6.中文模糊搜索：

在 C++中英文一个字符占一个字节，而中文一个字符占 3 个字节，故不能采用单纯的字符串匹配函数。我们在使用 string::find 的基础上增加模 3（%3）操作，判断搜索到的子串

第一次出现的下标是否为 3 的整数倍，如果是才可认为匹配成功。

该算法能解决绝大部分中文匹配异常的问题，但仍有样例会造成错误，如“人类”和“死”将被认定为相同字符串。不过由于我们的程序中不涉及错误字段，故可以信任该算法。

该算法应提前筛除匹配串为空串的情况，否则一定认为匹配成功：空串是任意串的子串，出现下标为 0。

7.模糊搜索结果选择

通过动态创建多个 QLabel 和 QPushButton，并绑定同一槽函数，使用 Lambda 表达式传值以确定所点击按钮的编号，实现将多个搜索结果展示于用户界面供用户选择。用一个指针数组记录动态创建的组件，以便在选择后全部删除释放空间。

8.路线显示问题

只有在当前画面显示的地图与当前导航所处地图相匹配的情况下才显示导航的线路。因此通过设置当前显示地图的标志，将之与当前状态所在地图进行匹配，若成功匹配则将当前位置的小圆点加入到 QGraphicsScene 中，并且将 QPathItem 的 path 设置为当前地图对应的 QPainterPath。如此可实现即使地图不匹配，图形化的移动仍会进行，只不过没有显示出来。

5.4 Clock 模块

时钟模块，提供模拟导航系统的时间。

主要函数及功能如下：

函数原型	功能
`QString get_clock_str()`	将时钟转化为格式化字符串并返回
`void update_clock()`	系统时钟更新，+1s
`int difSec(const Clock &clock)`	求两时钟相差的秒数
`bool operator>=(const Clock &clock)`	重载运算符>=：用于判断时间的大于等于关系
`Clock operator-(int h)`	重载运算符-：用于计算时间减去小时数得到的时间

5.5 Current 模块

当前状态模块，提供用户当前状态信息及周边信息。

主要函数及功能如下：

函数原型	功能
`bool inShahe()`	判断当前状态是否在沙河校区

`bool inBenbu()`	判断当前状态是否在本部校区
`bool inBetween()`	判断当前状态是否在校外地图

5.6 Dijkstra 模块

迪杰斯特拉算法模块，根据传入的优先度（距离/时间）提供两点间的最短距离（时间）路径的算法，并且结合 BFS 实现搜索一定范围内的所有顶点及其路径的功能

共实现四种功能：

1.单源点的两点寻路：

起点和终点均为顶点的最短路径算法。

2.双源点的两点寻路：

起点位于边的中间（不在顶点上），终点为顶点的最短路径算法。

3.单源点的范围搜索：

起点为顶点的 Dijkstra+BFS 算法。

4.双源点的范围搜索：

起点位于边的中间（不在顶点上）的 Dijkstra+BFS 算法。

主要函数及功能如下：

函数原型	功能
`void singleSearch(int begin, qreal r, qreal v)`	搜索单源点一定距离范围内的建筑并

	计算到各建筑的行进路线
`void doubleSearch(int begin1, int begin2, qreal r, qreal v, qreal ldis, qreal rdis)`	搜索双源点一定距离范围内的建筑并计算到各建筑的行进路线（当点位于某边而非节点上时，以边两端的节点作为源点）
`void singleFind(int begin, int end, Priority priority, qreal v)`	根据不同策略计算单源点到终点的行进路线
`void doubleFind(int begin1, int begin2, int end, Priority priority, qreal v, qreal ldis, qreal rdis)`	根据不同策略计算双源点到终点的行进路线（当点位于某边而非节点上时，以边两端的节点作为源点）
`void start()`	Dijkstra 算法
`void search()`	Dijkstra+BFS 算法
`void insert_to_visit(int x)`	插入待遍历顶点，保证顶点集有序

1.单源点两点寻路算法（Dijkstra 算法）描述：

（1）数据结构

```
QVector<int> to_visit;
```

待遍历顶点集，维持为一个距离（时间）递减的优先队列，每次遍历弹出其队头元素。

```
QMap<int,int> in_visited;
```

已处理的顶点集，Key 为顶点编号，Value 为 1 表示在集合内，否则不在。由于 in_visited 不需要遍历，而需要经常查询某个顶点是否已处理，故使用底层为红黑树的 QMap 容器，提高查询效率。

```
QMap<int,int> parent;
```

父顶点表，Key 为顶点编号，Value 为其对应的父顶点的编号。同样由于经常需要查询某顶点的父亲，故用 QMap 存储。该表用于逆向生成路径。

```
QVector<Vertex*> dij_path;
```

路径数组，值为顶点指针。

```
QMap<int,qreal> dis;
```

距离表，Key 为顶点编号，Value 为其到起点的距离。使用 QMap 原因如上。

```
QMap<int,qreal> time;
```

时间表，Key 为顶点编号，Value 为其到起点的时间。使用 QMap 原因如上。

（2）操作步骤

① 初始时，visited 为空，to_visit 中只有起点 s。

② 然后，从 to_visit 中弹出队头顶点（距离（时间）最短的顶点）记为 current，并将其加入到 visited 中。

③ 接着，遍历不在 visited 中的与 current 邻接的所有顶点，计算他们到 current 的距离 d（时间 t），如果比原先到该点的距离（时间）短，就更新它，并且将其 parent 设为 current。

④ 重复②③操作，直到所有顶点都已加入 visited。

最后通过 parent 表，从终点 end 逆向求出最短路径。

伪代码如下：

```
While（to_visit 非空）{
    Current = to_visit 队头;
    弹出 to_visit 队头;
    current 加入 visited;
```

```
        For(所有 current 的邻接点) {
            If(该点 in_visited) continue;
            If((优先度为距离优先 && 该点到 current 的距离
        +current 到起点的距离 < 该点到顶点的距离) ||
                (优先度为时间优先 && 该点到 current 的距离
    +current 到起点的时间 < 该点到顶点的时间)) {
                    更新 dis;
                    更新 time;
                    该点的 parent = current;
            }
            If（current 没有父亲） current 的 parent = begin;
        }
    }
    int p = end;
    while(p!=begin) {
        dij_path.push_front(vertices[p]);
        p = parent[p];
    }
    dij_path.push_front(vertices[begin]);
```

2.双源点两点寻路算法（Dijkstra 算法）描述：

分别从当前所在边的左端点（记为 lbegin）、右端点（rbegin）

进行单源点寻路 dij_left 和 dij_right。记当前位置到 lbegin 的距离为 ldis、时间为 ltime；到 rbegin 的距离为 rdis、时间为 rtime。

若 dij_left 的距离+ldis<dij_right 的距离+rdis(时间同理)，取 dij_left 路径为最短路径，并将 dij_left 的路径起点为 current 状态的下一顶点。

3.单源点范围搜索算法（Dijkstra+BFS 算法）描述：

数据结构与算法 1 类似，仅用如下数据结构替换 dij_path：

QMap<int,QVector<Vertex*>> paths：

路径表，Key 值为顶点编号，Value 为以该顶点为终点的最短路径。

伪代码如下：

```
While（to_visit 非空）{
    Current = to_visit 队头;
    弹出 to_visit 队头;
    if(dis[current]>搜索半径)  break;
    current 加入 visited;
    For(所有 current 的邻接点) {
        If(该点 in_visited) continue;
```

```
        If((优先度为距离优先 && 该点到 current 的距离
    +current   到起点的距离 < 该点到顶点的距离) ||
          (优先度为时间优先 && 该点到 current 的距离
+current   到起点的时间 < 该点到顶点的时间)) {
              更新 dis;
              更新 time;
              该点的 parent = current;
        }
        If（current 没有父亲） current 的 parent = begin;
    }
    逆向求出 current 的路径，加入路径表
    QVector<Vertex*> temp;
    int p = current;
    while(p != begin) {
        temp.push_front(vertices[p]);
        p = parent[p];
    }
    temp.push_front(vertices[p]);
    paths[current] = temp;
  }
```

5.7 Ant 模块

蚁群算法模块，提供途径多点的最短路径算法。

由于途经多点算法与传统的旅行商问题略有不同，因此蚂蚁最终不必回到起点，蚂蚁行进使用的完全图也并非初始邻接表，而是根据途经顶点集合由 Dijkstra 算法生成的（该操作在主窗口内完成，故此模块中不涉及）。

蚁群算法描述：

（1）基本原理

① 根据具体问题设置多只蚂蚁，分头并行搜索。

② 每只蚂蚁完成一次周游后，在行进的路上释放信息素，信息素量与解的质量成正比。

③ 蚂蚁路径的选择根据信息素强度大小（初始信息素量设为相等），同时考虑两点之间的距离，采用随机的局部搜索策略。这使得距离较短的边，其上的信息素量较大，后来的蚂蚁选择该边的概率也较大。

④ 每只蚂蚁只能走合法路线（经过每个地点 1 次且仅 1 次），为此设置禁忌表来控制。

⑤ 所有蚂蚁都搜索完一次就是迭代一次，每迭代一次就对所有的边做一次信息素更新，原来的蚂蚁死掉，新的蚂蚁进行新一轮搜索。

⑥ 更新信息素包括原有信息素的蒸发和经过的路径上信息素的增加。

⑦ 达到预定的迭代步数，或出现停滞现象（所有蚂蚁都选择同样的路径，解不再变化），则算法结束，以当前最优解作为问题的最优解。

（2）信息素及转移概率的计算

$\Delta\tau_{ij}$——本次迭代边（i，j）上的信息素增量；

$\Delta\tau_{ij}^{k}$——第 k 值蚂蚁在本次迭代留在边（i，j）上的信息素量；

Q——信息素增加强度系数：500

m——蚂蚁个数：58

Nc——最大迭代次数：100

ρ——信息素蒸发系数：0.75

$1-\rho$——持久性（残留）系数；

$P_{ij}^{k}(t)$——时刻 t 蚂蚁 k 由城市 i 转移到城市 j 的概率（转移概率）；

$tabu_{k}$——蚂蚁 k 的禁忌表。

τ——信息素浓度表

η——能见度表

α——信息素比例因子：0.8

β——能见度比例因子：4.2

$P_{ij}^{k}(t)$——计算公式：$\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}$

（3）算法步骤

① 初始化参数:开始时每条边的信息素量都相等。

$$\tau_{ij}(0)=C \quad \Delta\tau_{ij}(0)=0$$

② 将各只蚂蚁放置各顶点，禁忌表为对应的顶点。

③ 取 1 只蚂蚁，根据公式计算出转移概率 $P_{ij}^{k}(t)$，按轮盘赌的方式选择下一个顶点，更新禁忌表，再计算概率，再选择顶点，再更新禁

忌表，直至遍历所有顶点 1 次。

④ 计算该只蚂蚁留在各边的信息素量$\Delta\tau_{ij}^{k}$，该蚂蚁死去。

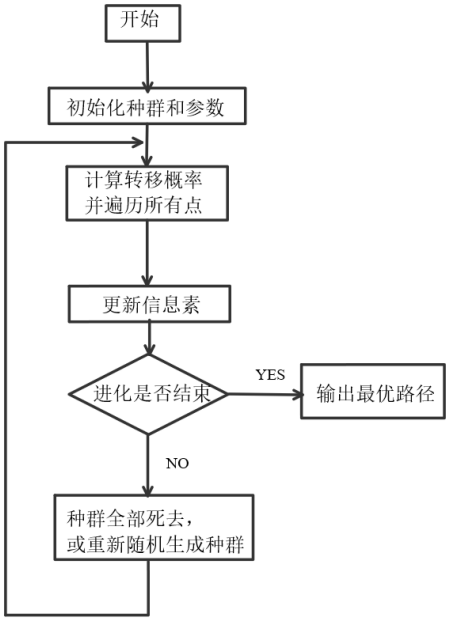
⑤ 重复③④，直至 m 只蚂蚁都周游完毕。

⑥ 计算各边的信息素增量$\Delta\tau_{ij}$，和信息素量$\tau_{ij}(t+n)$。

⑦ 记录本次迭代的路径，更新当前的最优路径，清空禁忌表。

⑧ 判断是否达到预定的迭代步数，或者是否出现停滞现象。若是，算法结束，记录当前最优路径及距离、时间；否，转②，进行下一次迭代。

算法流程图如下：



主要函数及功能如下：

函数原型	功能
`void start(const QVector<int> &spots, Priority priority)`	蚁群算法开始迭代，参数分别为顶点集和优先

	度（距离/时间）
`void clearAll()`	清空所有辅助表（ph、visibility 等）
`void shuffle()`	洗牌，随机取一个顶点作为蚂蚁的起点
`void move()`	蚂蚁根据概率，结合轮盘赌进行移动
`void record()`	记录单次迭代的最短路径、距离、时间等
`void record_inall()`	记录所有迭代中的最短路径、距离、时间等
`void update()`	更新每条路径的信息素浓度
`void init_ph()`	初始化信息素浓度表
`void init_delta_ph()`	初始化信息素增量表
`void init_visibility()`	初始化能见度表
`void init_tabu()`	初始化禁忌表
`void clear_tabu()`	清空禁忌表
`qreal rnd(qreal lower, qreal upper)`	返回 lower 到 upper 中的一个随机数

5.8 LogSearch 模块

日志查询模块，提供查询日志文件相关信息的功能。

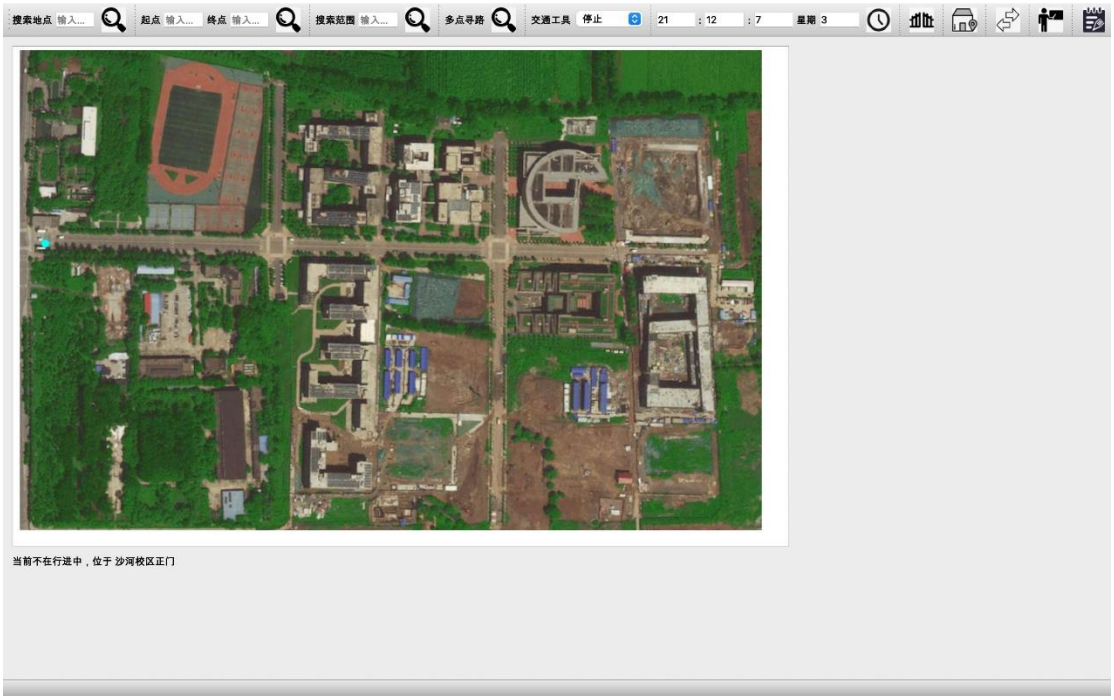
主要函数及功能如下：

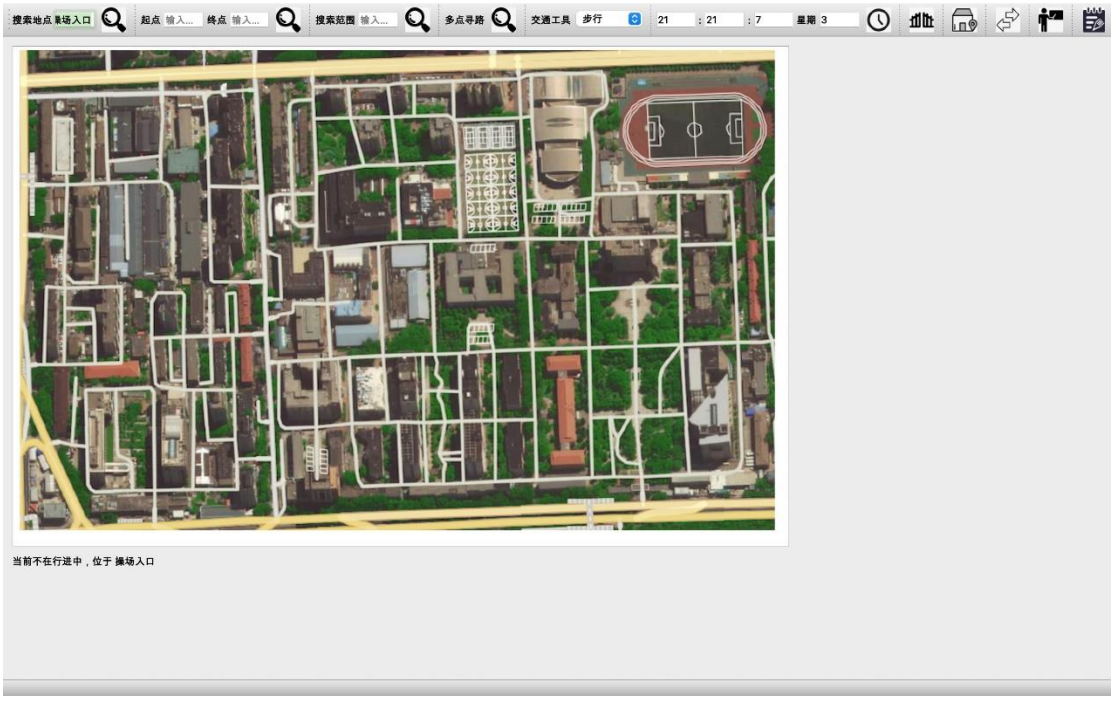
函数原型	功能
`void sendData(Clock cur)`	主窗口传入当前时间
`void on_pushButton_clicked()`	查询按钮的槽函数，根据操作和时间筛选日志记录并输出

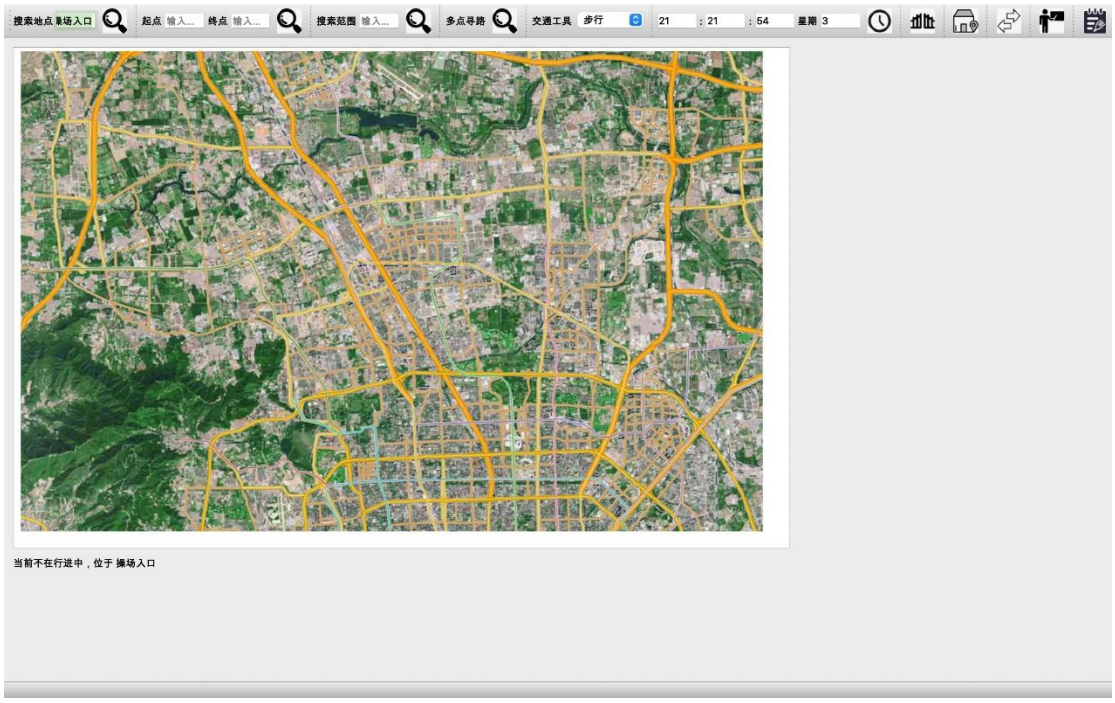
六、测试样例

1、地图更换

分别是沙河地图、本部地图、校外地图



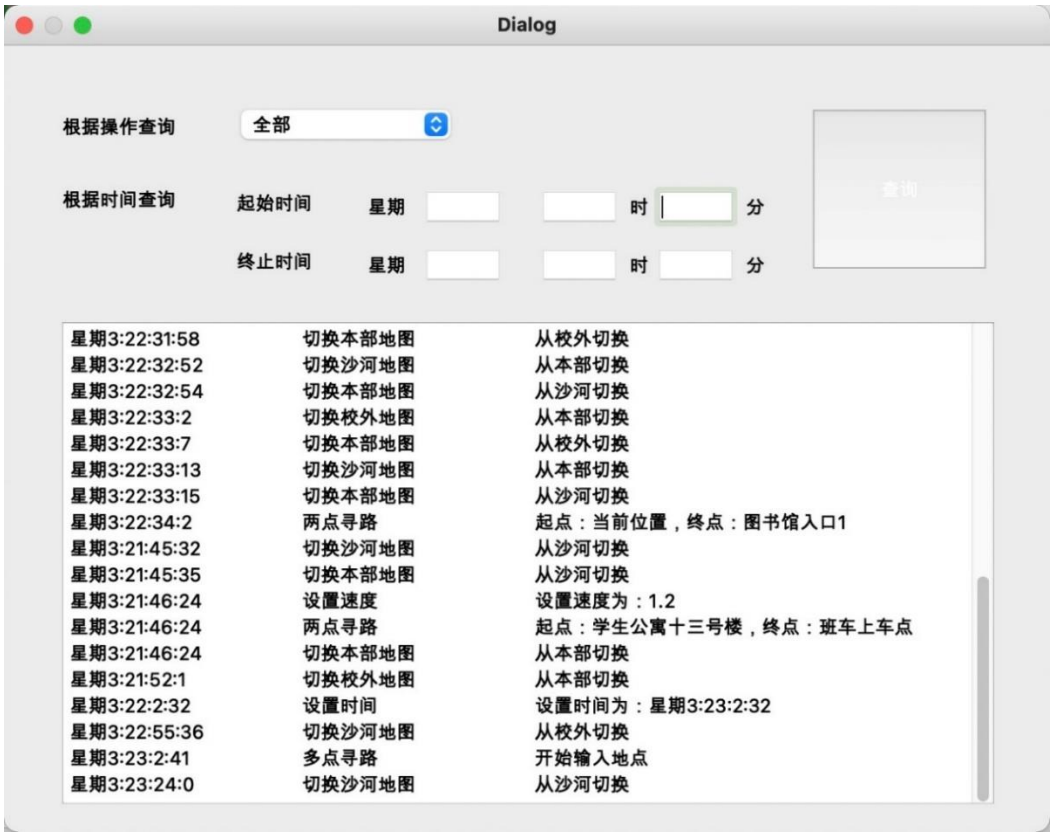




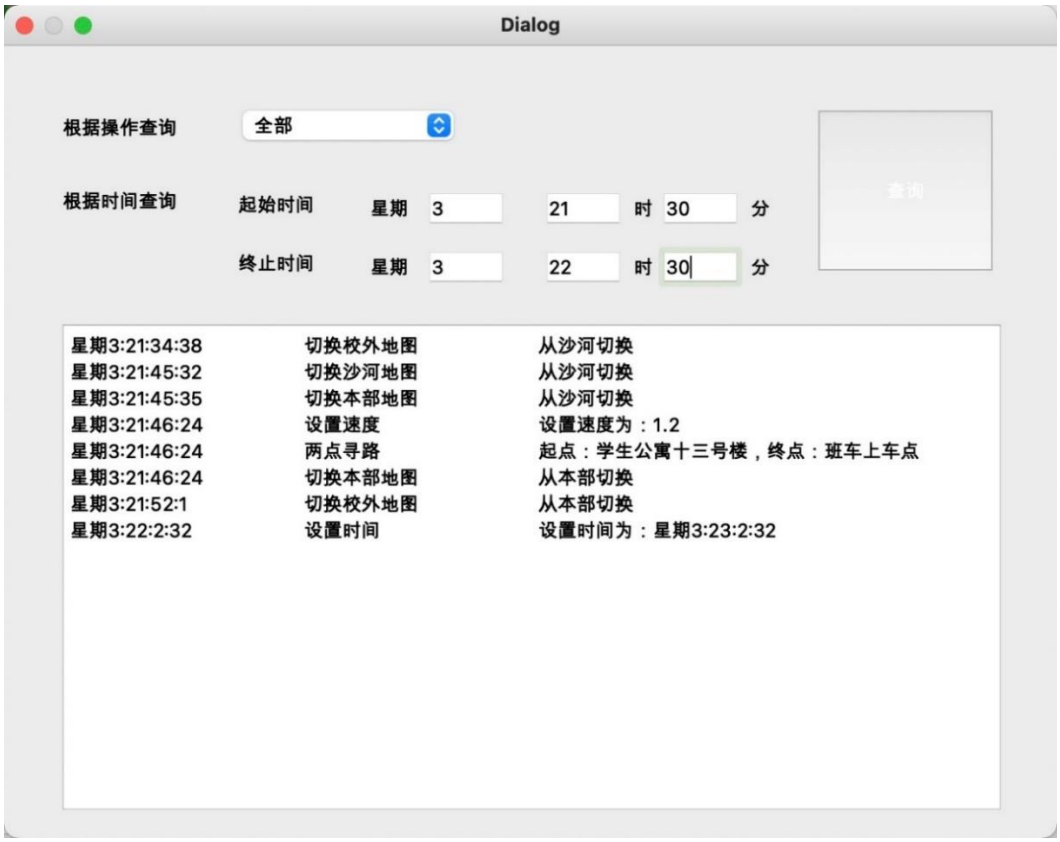
2、更换当前出行方式

3、日志查询

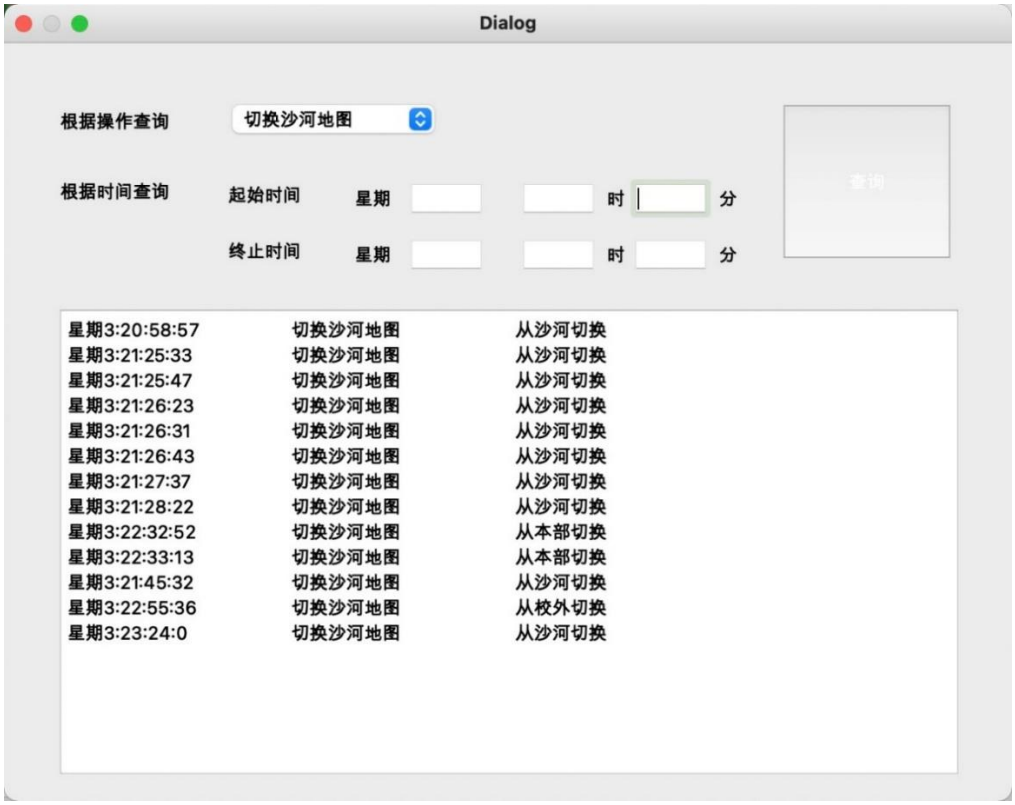
1)默认查询前三个小时日志



2) 自行设置查询起止时间



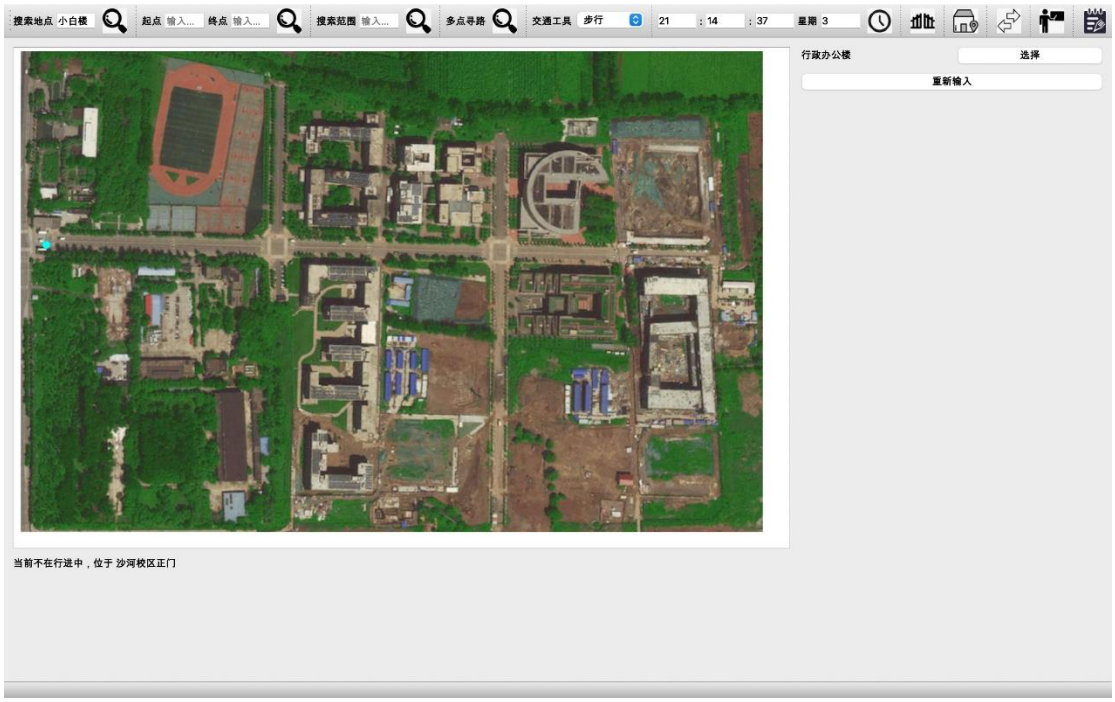
3)根据操作查询



4、查询功能

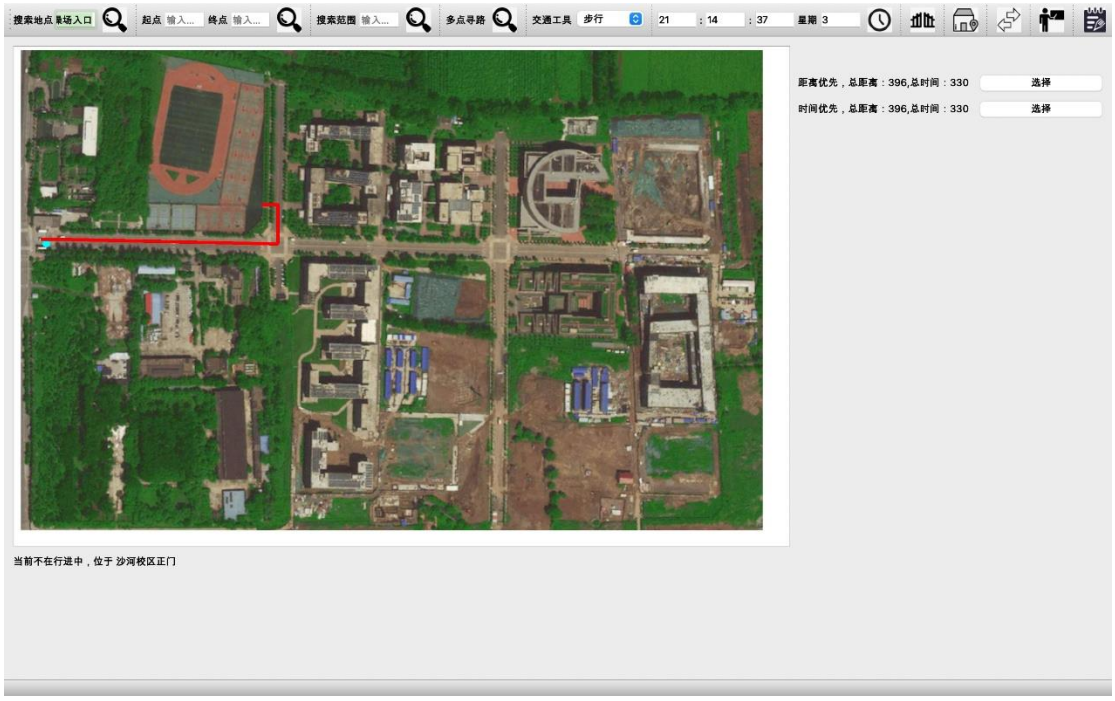
1)全部搜索栏支持模糊搜索

2)全部搜索栏支持逻辑名搜索

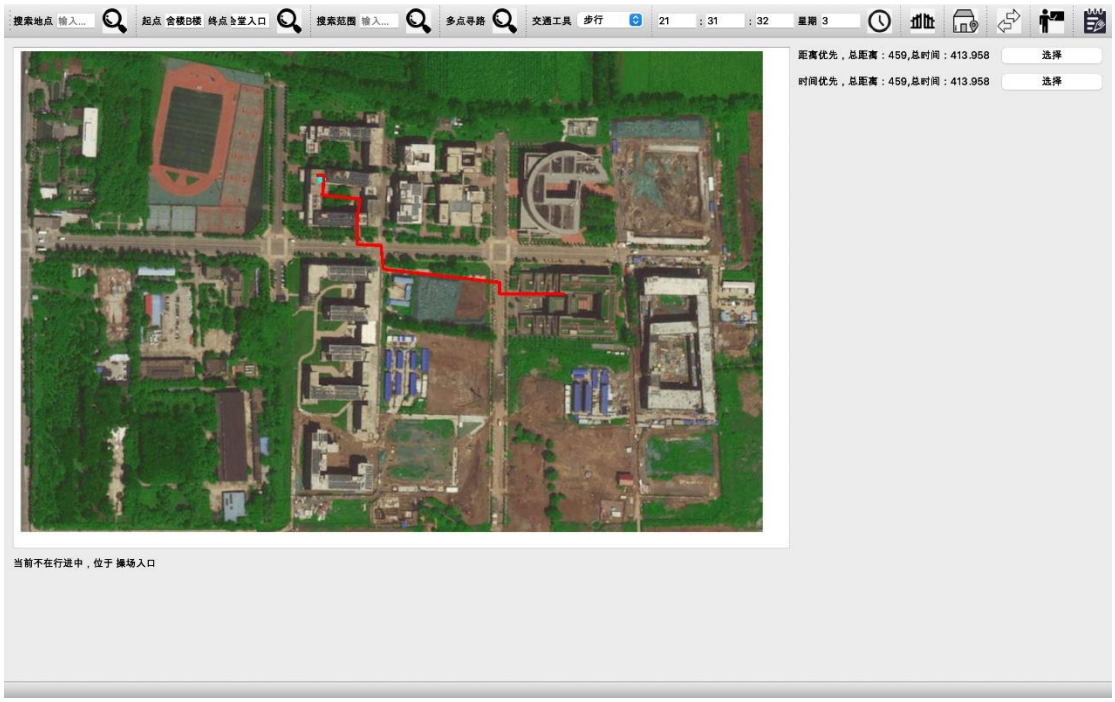


5、导航

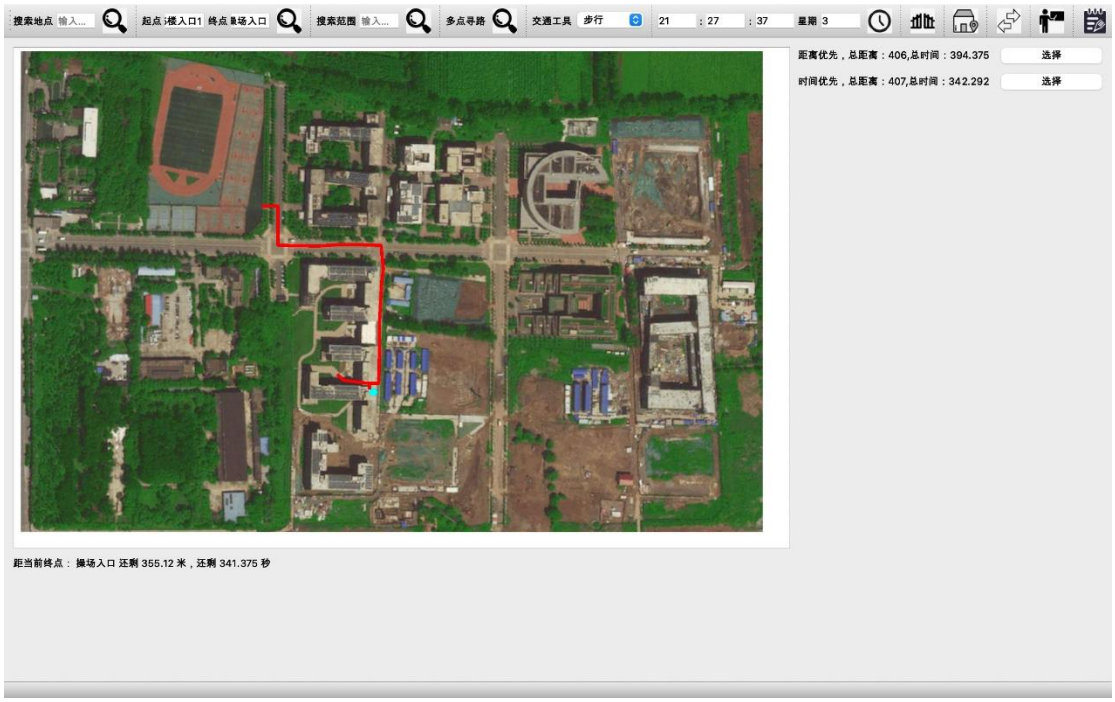
1)指定终点寻路

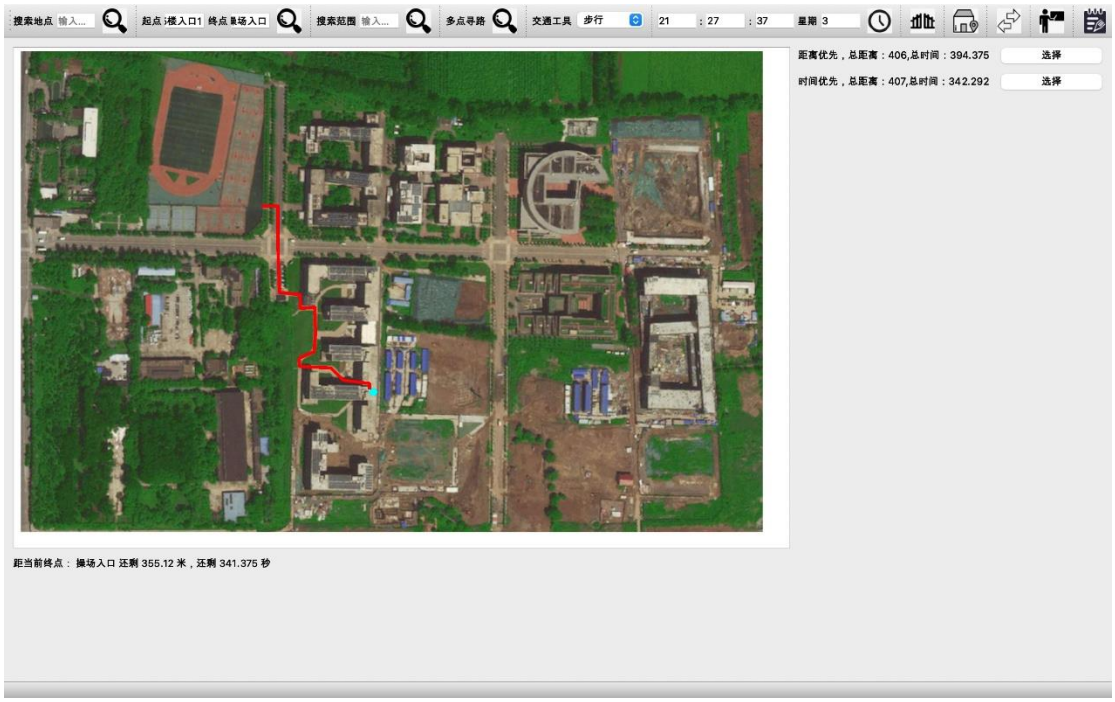


2)用户指定起止点寻路



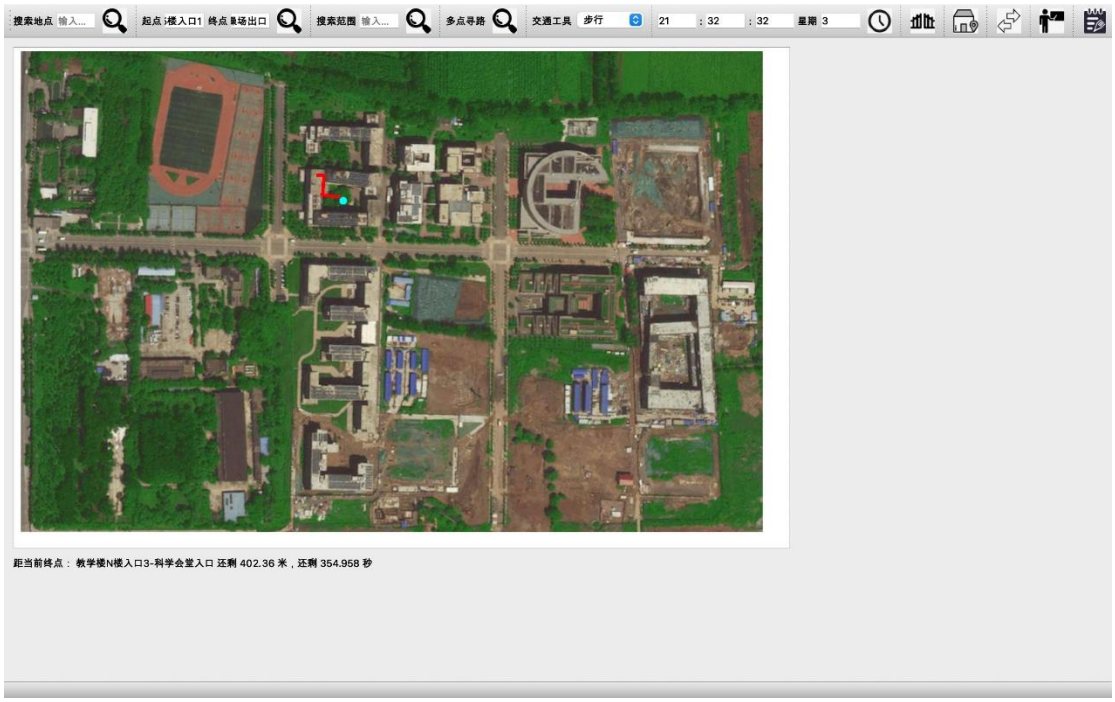
3)用户选择不同的优先级会有不同的路径



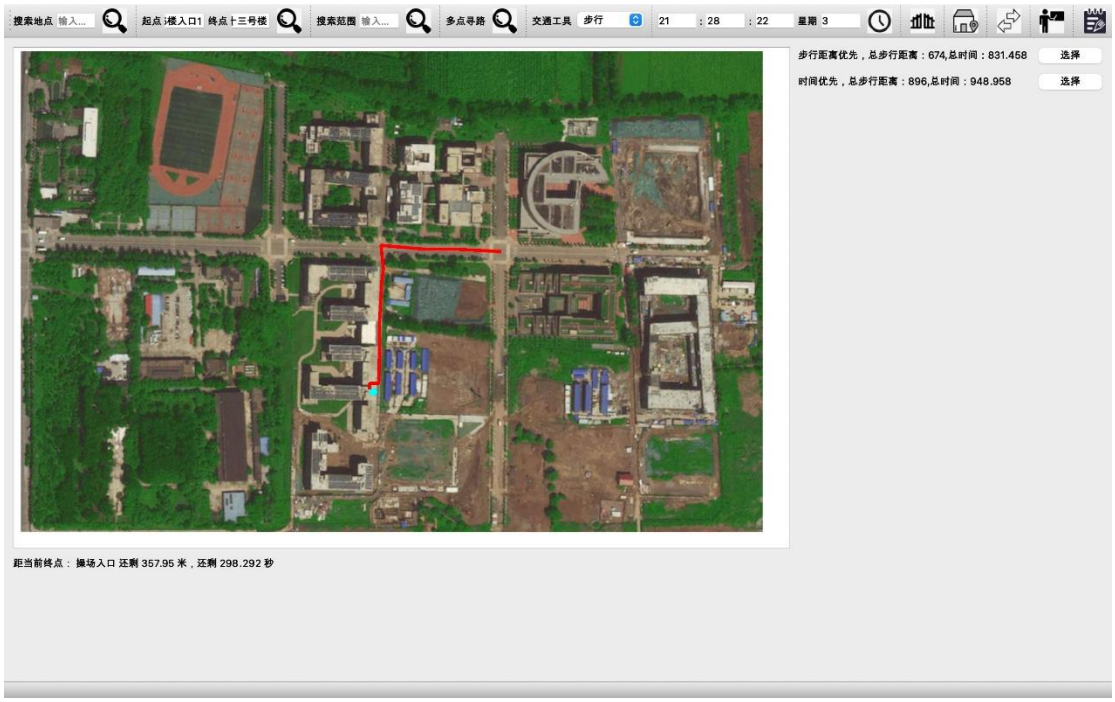


4) 寻路完成后开始导航并不断更新当前状态

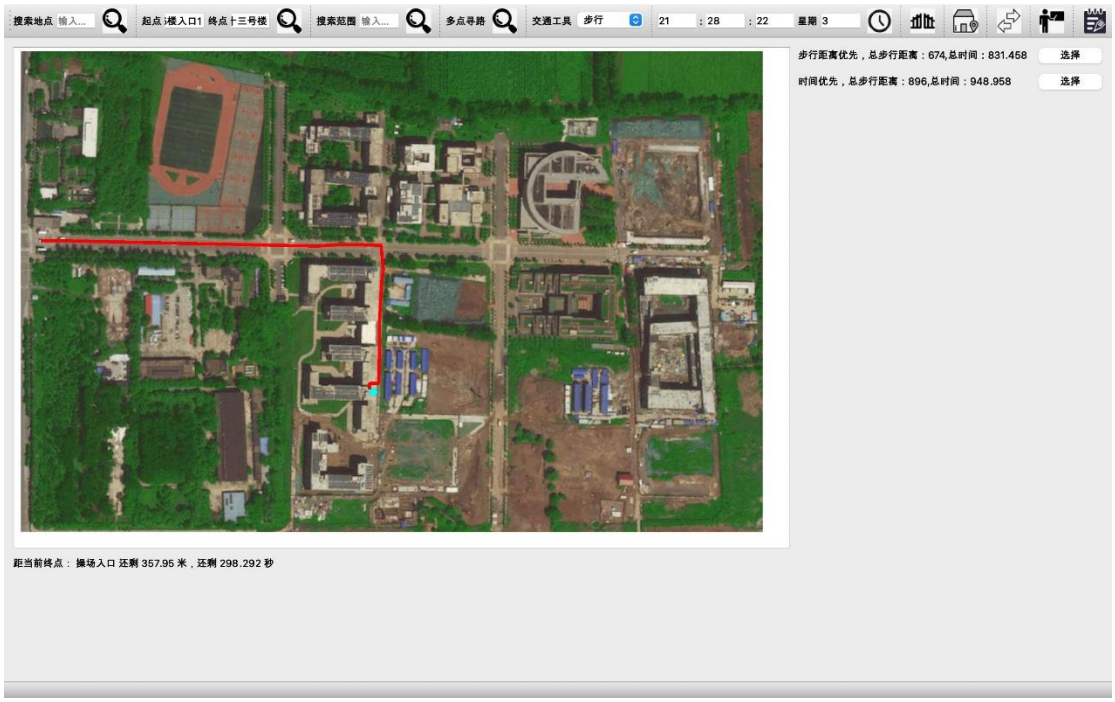
5)导航过程中进行操作系统会自动暂停



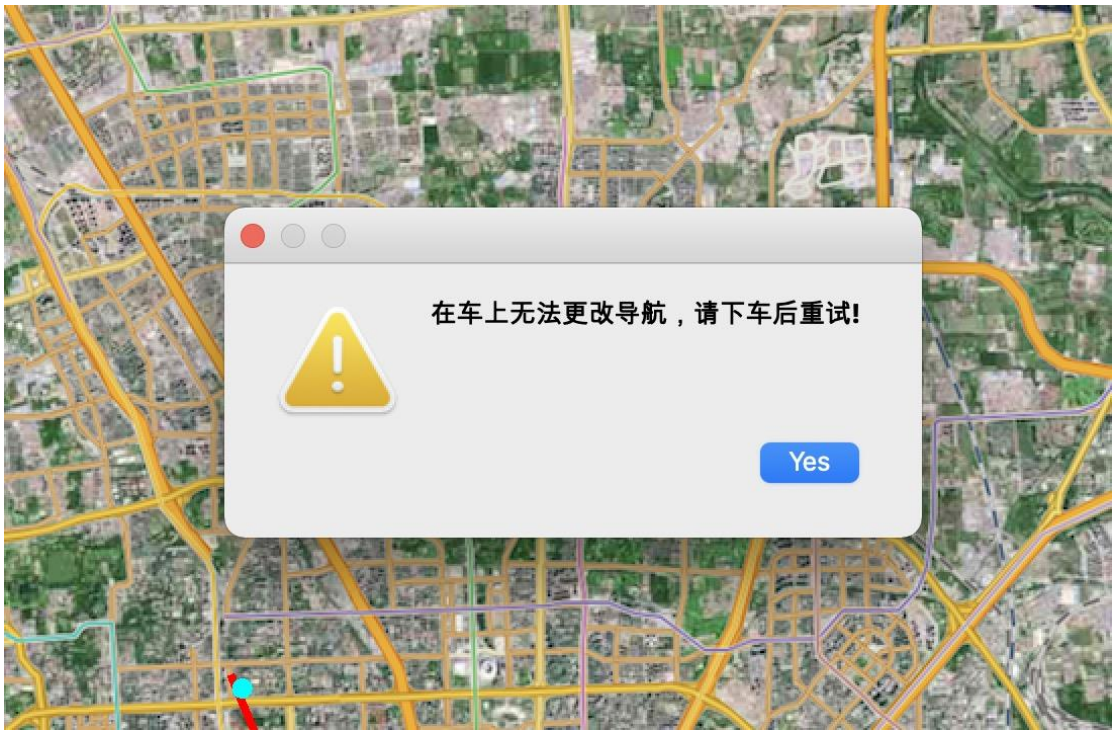
6)跨校区两点导航(优先展示第一段导航路径)



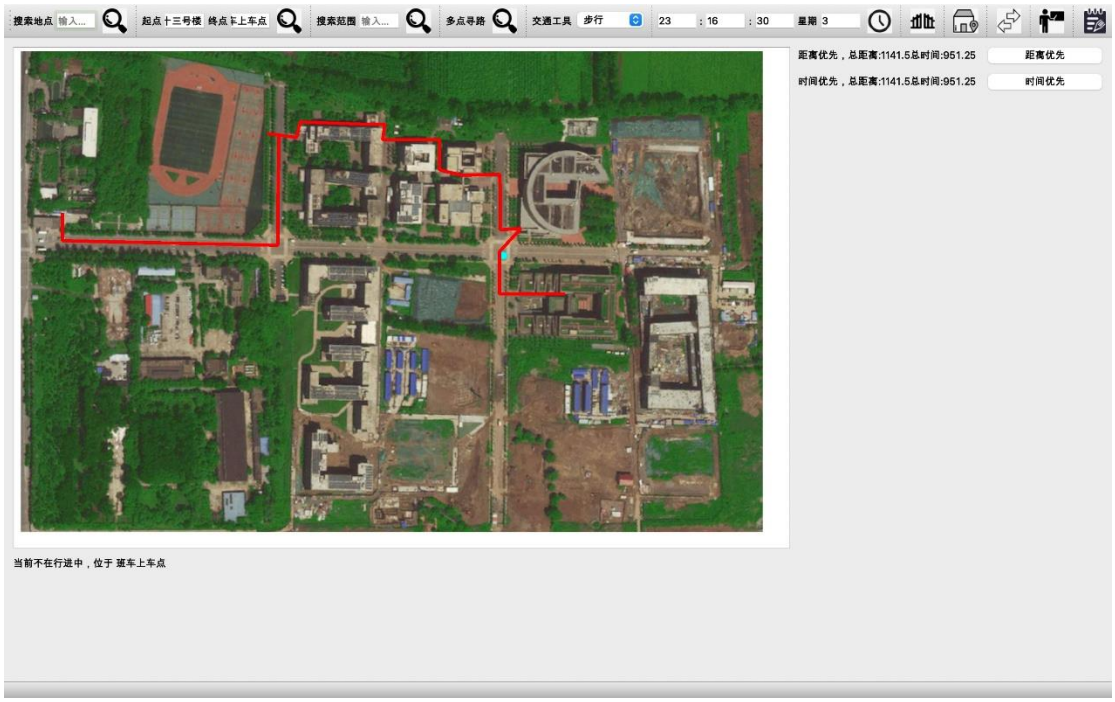
系统自行判断不同优先级的时候的不同路径



7) 当试图在班车上或者公交上更改导航逻辑



8)多点寻路(沙河-图书馆、快递站操场出口、教学楼)



七、评价及改进建议

7.1 总体概述

本校园导览系统基本实现了校园内的导航，支持两个校区的导览，教学楼、图书馆、宿舍、体育馆等近 50 个建筑，360 个顶点，近 700 条边，并可以查询到校园内各种建筑物及服务设施的相关信息。在导航时，支持最短距离、最短时间、途径最短距离及交通工具的最短时间等四个策略，通过图形化界面实现了模拟导航，在导航时用户可以实时查询周边建筑信息，并可设置当前模拟导航的时间速率。本系统可以记录当前运行情况的日志文件，如学生状态变化和键入信息。

7.2 优点

1)本系统实现了图形化界面，增强了和用户之间的交互性，使用起来更加便捷直观。系统界面各功能清晰，操作逻辑性强。

2)用户可以直接在本图形化界面上进行双击操作以查看相关点的信息。

3)本校园导览系统可以实时在地图上进行导航，并更新当前状态信息，在行进途中可以根据设置的通行率来判断最短时间的路径，且在跨校区导航时会自动计算等车时间，并根据优先度选择不同的跨校区方式。

4)本系统实现了在行进途中可自行更改目标和导航策略，系统按照当前位置做相应操作。

5)用户在进行搜索时，可以实现模糊搜索以及逻辑名和实际地址名的对应，让用户不再担心自己会不会无法搜索到自己想去的目的地。

6)此外，本系统根据时钟判定当前所在时间及星期，自行匹配账户课程表中的当前课程信息并进行导航。

7)本系统内置时钟，基本默认模拟速率为 167ms 模拟为现实的 1s，并可以进行模拟速率的更改，在满足模拟的同时也不会耗费太多的时间。

7.3 改进建议

1)本系统在人流量判断负载的方面并没有做到很好的层次，未能

实现负载均衡。

2)本系统暂时只是实现了由课程表到建筑物的对应，并没有继续实现到对应楼层及房间的对应，相同的是，对于宿舍楼内部的具体实现当前也抱憾未能实现。

八、用户手册

具体见附件用户使用说明书。

附录：源代码

具体见附件。