

实验报告



题 目： 校园导览系统——数据结构课程设计

班 级： 2019211319

学 号： 2019211865；2019212098；2019211898

姓 名： 陈伟杰；王博荣；王天一

学 院： 计算机学院

2021年 6 月 1 日

目录

[一、软件开发任务的描述 4](#_Toc73740020)

[二、功能需求说明和分析 4](#_Toc73740022)

[1、 基础导航功能 4](#_Toc73740023)

[2、 不同导航策略的选择 4](#_Toc73740026)

[3、 模拟导航 5](#_Toc73740028)

[4、 查询周边环境信息 5](#_Toc73740030)

[5、 日志保存 5](#_Toc73740032)

[6、 实时导航 5](#_Toc73740034)

[7、 均衡负载 6](#_Toc73740036)

[三、总体方案设计说明 6](#_Toc73740037)

[1、软件开发环境： 6](#_Toc73740038)

[2、总体结构流程图 7](#_Toc73740039)

[3、模块划分 8](#_Toc73740040)

[四、数据结构说明和数据字典 10](#_Toc73740041)

[1、Form类（主界面） 10](#_Toc73740042)

[2、querywidget类（从界面） 11](#_Toc73740043)

[3、recordwidget类（日志页面） 11](#_Toc73740044)

[4、Widget类（欢迎界面） 12](#_Toc73740045)

[5、map模块（地图模块） 12](#_Toc73740046)

[1、 地图结点结构体： 12](#_Toc73740047)

[2、 全局变量： 12](#_Toc73740048)

[3、 函数： 12](#_Toc73740049)

[6、navigation （链路规划模块） 13](#_Toc73740050)

[1、 结构体： 13](#_Toc73740051)

[2、 函数： 13](#_Toc73740052)

[7、findTheWay （路由规划模块） 13](#_Toc73740053)

[五、模块设计说明 14](#_Toc73740054)

[程序模块总述 14](#_Toc73740055)

[1、 main初始化模块 16](#_Toc73740056)

[2、 map地图载入模块 17](#_Toc73740057)

[3、 widget UI样式设置模块 21](#_Toc73740058)

[4、 form 主窗口初始化模块 22](#_Toc73740059)

[5、 querywidget 从窗口初始化模块 22](#_Toc73740060)

[6、 buttonEvent点击事件处理模块 24](#_Toc73740061)

[7、 Recode 日志模块 28](#_Toc73740062)

[8、 selectCanteen 食堂均衡模块 28](#_Toc73740063)

[9、 findTheWay 路由规划模块 29](#_Toc73740064)

[10、 navigation 路线导航模块 32](#_Toc73740065)

[11、 aboutTime 时钟模块 35](#_Toc73740066)

[12、 IOfunction IO模块 / QIchange 字符串转换模块 36](#_Toc73740067)

[六、 应用范例执行结果及测试情况说明 37](#_Toc73740068)

[1、 途径最短距离： 37](#_Toc73740069)

[2、 跨校区导航： 40](#_Toc73740070)

[六、评价和改进意见： 41](#_Toc73740071)

[七、用户使用说明： 42](#_Toc73740072)

# **一、软件开发任务的描述**

本软件设计实现了一个校园导览系统，基于数据结构课程知识实现多方式寻路、模糊输入处理、路途周边查询，以及实时变更策略、均衡负载等功能。此外，为了向用户提供简洁直观的应用窗口，开发内容还包括实现图形化界面与动态效果、错误输入提醒等交互内容。

# **二、功能需求说明和分析**

## **基础导航功能**

校园导览系统的核心功能即为分析地图与用户起点终点，给出最合适的道路规划，同时本系统创新性地采用了新的选点方式以便更精准地确定路线，故对此核心算法进行展开详述。系统采用Dijkstra算法，选取①所有道路交叉点与②各建筑物的出口及其正对道路的路边，这两类地点作为图的节点，规定各节点的邻接结点为：该点在地图上仅通过一条道路即可到达的结点，距离即空间实际距离。

为了支持用户途经点的需求，系统采用

## **不同导航策略的选择**

考虑到为用户提供不同情况下的最优路径规划，如步行、骑行、时间敏感、距离敏感四种情况下的不同规划，系统对时间、拥挤度采用了加权赋值给距离，使得地图所保存的空间距离在经过复合后以“广义距离”的形式参与路径选取计算。

## **模拟导航**

考虑到实际行走时间过长不便演示，系统采用对外提供倍速选择接口的形式，可以按照模拟需求进行不同程度的加速，同时也可以按实际情况进行跟踪。

## **查询周边环境信息**

在行走过程中，为向用户提供除行走方向之外的其他的信息，系统维护了一份与建筑相关的信息事件，并与当前位置进行实时比对，若距离小于某一固定值时即向用户显示。

## **日志保存**

为实时记录学生状态变化并键入信息，程序进行实时变更指令的保存。

## **实时导航**

考虑到用户可能存在实时变更策略与目标的需求，系统可以随时将当前位置赋予起始点，与新目标或新策略调用寻址函数进行新的导航。

## **均衡负载**

以食堂的均衡负载为例。在程序运行过程中，当用户选择前往食堂，系统将基于当前运行软件的用户日志文件进行食堂人流量统计与未来预测。若终点是人数较多的食堂时，系统将引导用户进行另外的选择。

# **三、总体方案设计说明**

## 1、软件开发环境：

操作系统：建议为Windows10（支持跨平台开发）

开发框架：Qt 5.3.1版

版本控制：Git

编程语言：C++，基于支持C++11及以上标准的版本

源文件编码：UTF-8（其他编码可能导致中文注释出现乱码的现象）

## 2、总体结构流程图

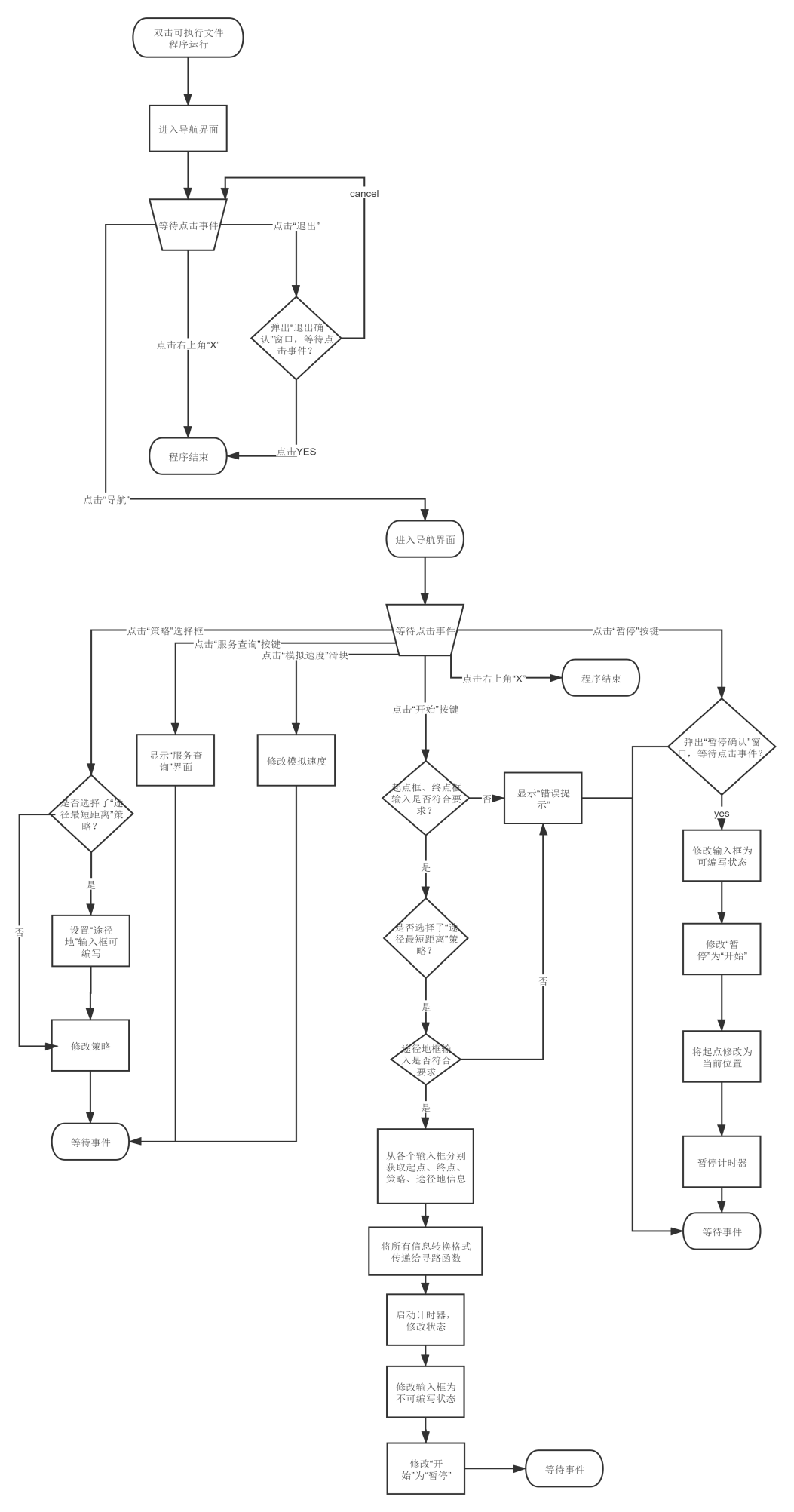


图1 程序总体框架和程序流程图

## 3、模块划分

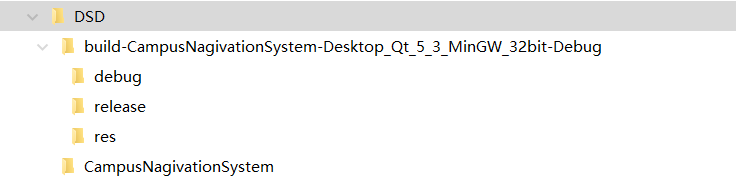


图2 文件夹结构图

|  |  |
| --- | --- |
| 文件 | 说明 |
| DSD | 总文件 |
| debug | 对象文件（.o）以及可执行文件（.exe） |
| res | 存放图片（.png/.jpg）和文本（.txt） |
| CampusNagivationSystem | 头文件（.h）、源文件（.cpp）和软件界面文件（.ui） |

表1 文件夹说明

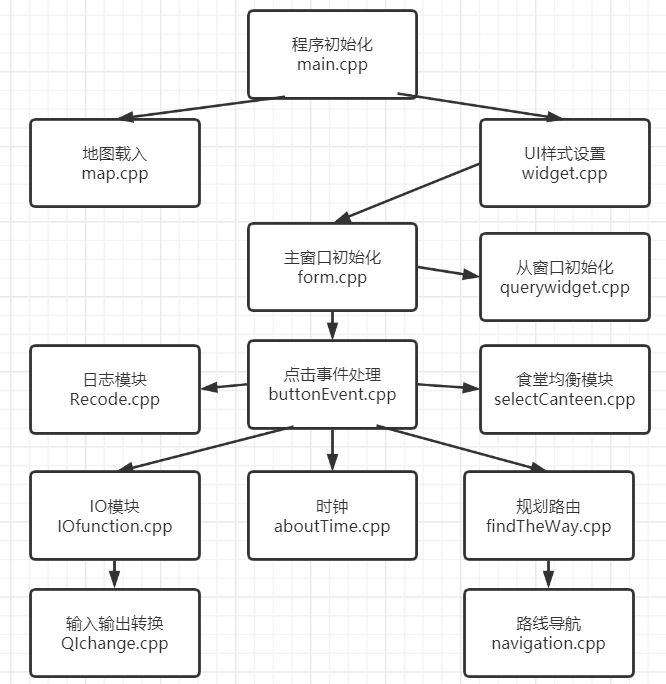


图3 模块划分和调用关系

# 四、数据结构说明和数据字典

## 1、Form类（主界面）

*public:*

*explicit Form(QWidget \*parent = 0);//构造函数*

*void paintEvent(QPaintEvent \*);//背景图设置函数*

*QueryWidget \*query = NULL;//从页面*

*~Form();//析构函数*

*int origin;//出发地*

*int terminal;//目的地*

*int strategy;//导航策略*

*double actualDistance;//实际距离*

*double formalDistance;//形式距离*

*std::vector<int> transfer;//途经地*

*int pos2int(QString position);//将地点从qstring类型转成int类型*

*QString int2pos(int positionNumber);//将地点从int类型转成qstring类型*

*void insertRecord();//日志函数*

*void selectCanteen();//食堂负载函数*

*RecordWidget recWid; //日志窗口*

*private slots: //Qt槽函数，当界面上控件状态改变时被触发*

*void on\_comboBoxStrategy\_currentIndexChanged(int index); //策略选择*

*void on\_pushBUttonStart\_clicked(); //点击“开始”*

*void on\_horizontalSlider\_valueChanged(int value); //模拟倍数*

*private:*

*Ui::Form \*ui; //图形化界面操作对象，可获取界面中的输入*

*int position;//当前位置*

*double leftTime;//目前抵达终点所需的时间*

*double speed;//前进速度*

*std::vector<int> pass\_position;//经过的路径点集*

*std::vector<int> distance\_of\_change;//剩余距离为何值时速度应该改变*

*std::vector<double> local\_speed;//在当前位置的前进速度*

*std::vector<int> getTransfer(QString transfer);//获取UI界面中的的途径地*

*QTimer \*mytime; //时钟对象*

*void changeStation();//按照时钟函数被调用，修该状态*

*void displayStation();//图形显示函数*

## 2、querywidget类（从界面）

*public:*

*explicit QueryWidget(QWidget \*parent = 0); //构造函数*

*void paintEvent(QPaintEvent \*); //设置背景图*

*~QueryWidget(); //析构函数*

*void luxian\_show(std::vector<int> t);*

*private:*

*Ui::QueryWidget \*ui; //图形化界面操作对象，可获取界面中的输入*

## 3、recordwidget类（日志页面）

*public:*

*explicit RecordWidget(QWidget \*parent = 0); //构造函数*

*~RecordWidget(); //析构函数*

*private slots:*

*//Qt槽函数*

*void on\_pushButtonClean\_clicked(); //清除日志*

*void on\_pushButton\_clicked(); //刷新日志*

*private:*

*Ui::RecordWidget \*ui; //UI界面操作对象，可获取界面中的输入*

## 4、Widget类（欢迎界面）

*public:*

*explicit Widget(QWidget \*parent = 0); //构造函数*

*void paintEvent(QPaintEvent \*);//背景图设置函数*

*Form \* wNag = NULL;//主窗口*

*~Widget(); //析构函数*

*private:*

*Ui::Widget \*ui; //UI界面操作对象，在本窗口中几乎无用*

## 5、map模块（地图模块）

### 地图结点结构体：

*typedef struct{*

*char node\_name[33]; // 节点名称,不超过31个字符*

*int next\_node\_number[6]; // 邻接节点的编号列表*

*int next\_node\_distance[6]; // 到邻接节点的距离*

*int next\_node\_crowd[6]; // 两节点间道路的拥挤程度*

*int next\_node\_bicycle[6]; //是否可以通行自行车，1为能，0为不能*

*} Node;*

*宏定义：*

*#define Number\_Total //地图总结点数*

*#define Number\_Xi //西土城校区结点数*

### 全局变量：

*static Node map[Number\_Total]; //地图结点数组*

*static int distanceBetweenPoint[Number\_Total][Number\_Total]; //最短距离矩阵*

### 函数：

*int InitMap(); //载入地图*

*void warshall(); //计算任意两点间的最短实际距离*

*void sort\_map(int map\_len); //地图排序*

*void save\_sorted\_map(QString selected\_map\_name, int map\_len); //保存地图数据*

## 6、navigation （链路规划模块）

### 结构体：

*typedef struct NaviTreeNode{*

*int FatherNode; //父节点代号*

*int formalDisFromStart; //到起点的形式距离*

*int actualDisFromStart; //到起点的实际距离*

*} NaviTreeNode; //结果树结点*

### 函数：

*bool NotFound(int num, int index, int HaveFound[]); //判断结点是否已经被找到*

*std::vector<int> navigation(int start, int end, int strategy)；//根据策略返回最佳路径*

## 7、findTheWay （路由规划模块）

*int insert\_gate(std::vector<int> \*points, int strategy, int XS, Form\* ui, int point\_num); //加入校车/公交车等待时间*

*std::vector<int> join\_path(std::vector<int> points, int strategy, double& actualDistance); //将多段路由拼合成一段*

*int shortest\_passing(std::vector<int> \*points, int strategy, int start\_index, int end\_index); //路由排序函数*

*bool is\_station(int one); //判断该结点是否是车站*

# 五、模块设计说明

## 程序模块总述

软件包括前端(GUI)部分和后端(算法)部分，共分为13个模块，分别为时钟模块（aboutTime）、点击事件处理模块（buttonEvent）、日志模块（Recode.cpp）、食堂均衡负载模块（selectCanteen.cpp）、路由模块（findTheWay）、导航模块（navigation）、主窗口设置模块（form）、IO模块（IOfunction）、初始化模块（main）、地图载入模块（map）、字符串转换模块（QIchange）、从窗口设置模块（querywidget）、UI样式设置模块（widget）。

各模块关系如下：

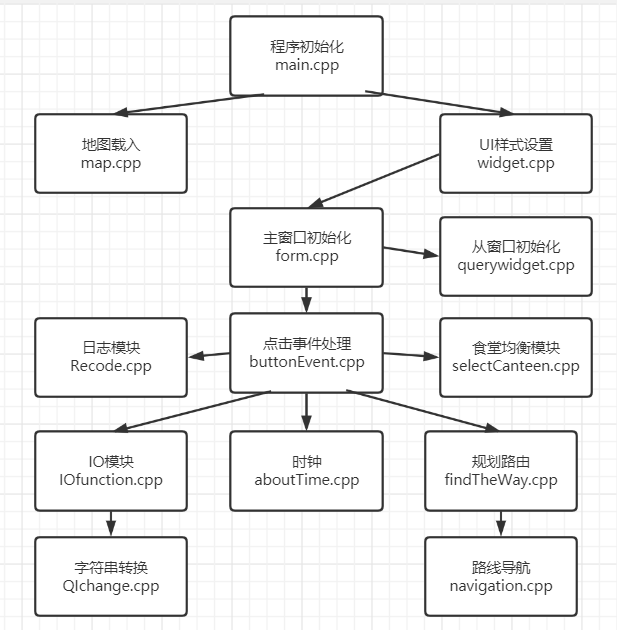


图1 模块间关系

main模块负责将地图载入结构数组中，并开启UI样式的初始化设置，UI样式设置完成后，等待用户点击进入程序。进入程序后，主窗口显示，同时完成从窗口的初始化，而后由buttonEvent模块处理各种点击事件，buttonEvent模块再调用IO模块、时钟模块、路由规划等模块，完成导航功能。

以下将介绍各模块的具体设计。

## main初始化模块

main模块的职能非常简单，只是调用其他模块的初始化函数，并设置地图拥挤度和自行车通行状况。

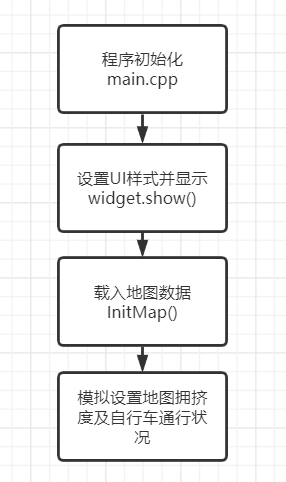


图1.1 main模块流程图

其中widget.show()是初始化窗口的显示函数，UI样式的设置在该窗口的构造函数完成。InitMap()为map模块中定义的初始化函数，能将文件中的地图数据载入到全局变量中，以供计算使用。

## map地图载入模块

本模块最大的特点就是将十字路口抽象成为节点的构图方式。采用沃舍尔算法构建全地图的最短路径，用于周边查询。

每一幢建筑都必定有与其大门相连的道路存在，而道路两边连通的地点具有固定性。倘若将十字路口看作节点，则相邻两个节点之间有且只有一条不经过其他节点的道路，以此为基础即可将整张图转变为一张简单无向图。如下图所示。



图2.1 具体地图的抽象化

每个节点只需记住其所有邻据节点的名字以及与邻居节点之间的距离即可还原出整个节点网络的拓扑结构。再加入其他附加信息，如拥挤度、是否能够通行自行车等，便足够满足本项目对地图的所有需求。故据此设计节点结构体如下：



图2.2 节点结构体

将地图节点数组设计为全局变量，以便所有函数查阅。考虑到寻路优化时的需要，额外设立了距离邻接矩阵，用于存储任意两点间的最短距离。

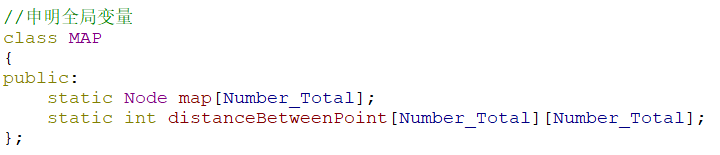
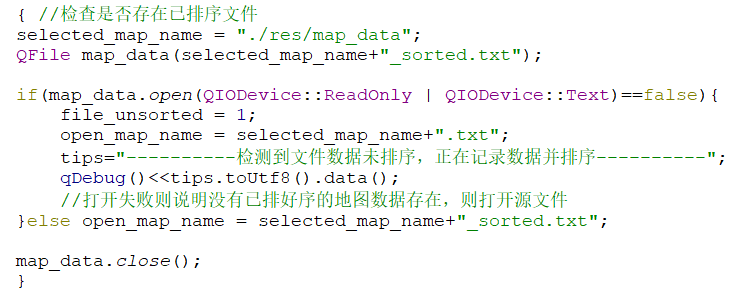
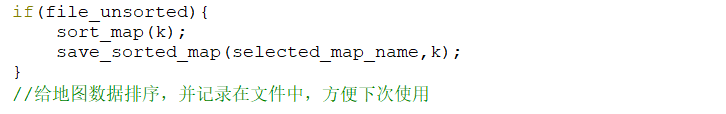


图2.3 全局变量申明

地图节点数组通过文件IO的方式进行初始化。为减小寻路时的开销，地图初始化时将会自动检查原始地图数据文件是否有序，若无序则会对其进行排序，而后生成新的数据文件。







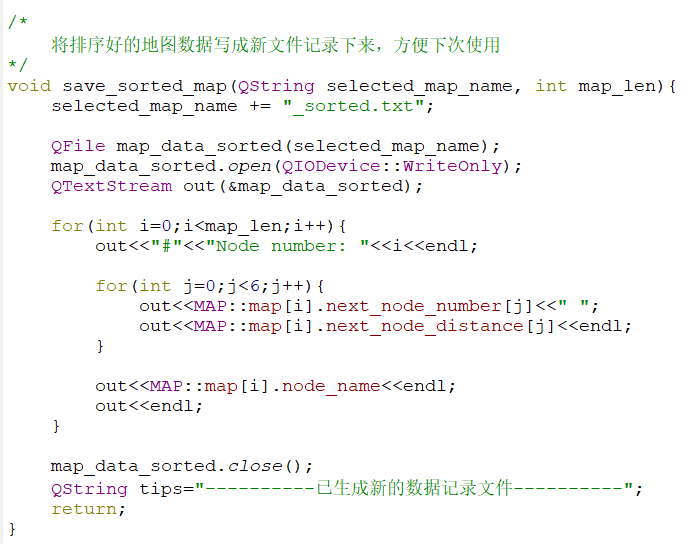


图2.4 检查、排序和保存

初始化函数需要传入参数，返回值为一个整数，用于告知主函数地图初始化是否成功。

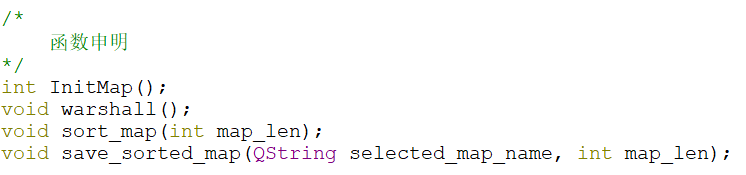


图2.5 模块函数所需参数及返回值

如上图所示，warshall函数没有返回值，采用沃舍尔算法计算图中每两点间的最短实际距离，在周边查询时可用方便地筛选出在用户附近的建筑物或路口。

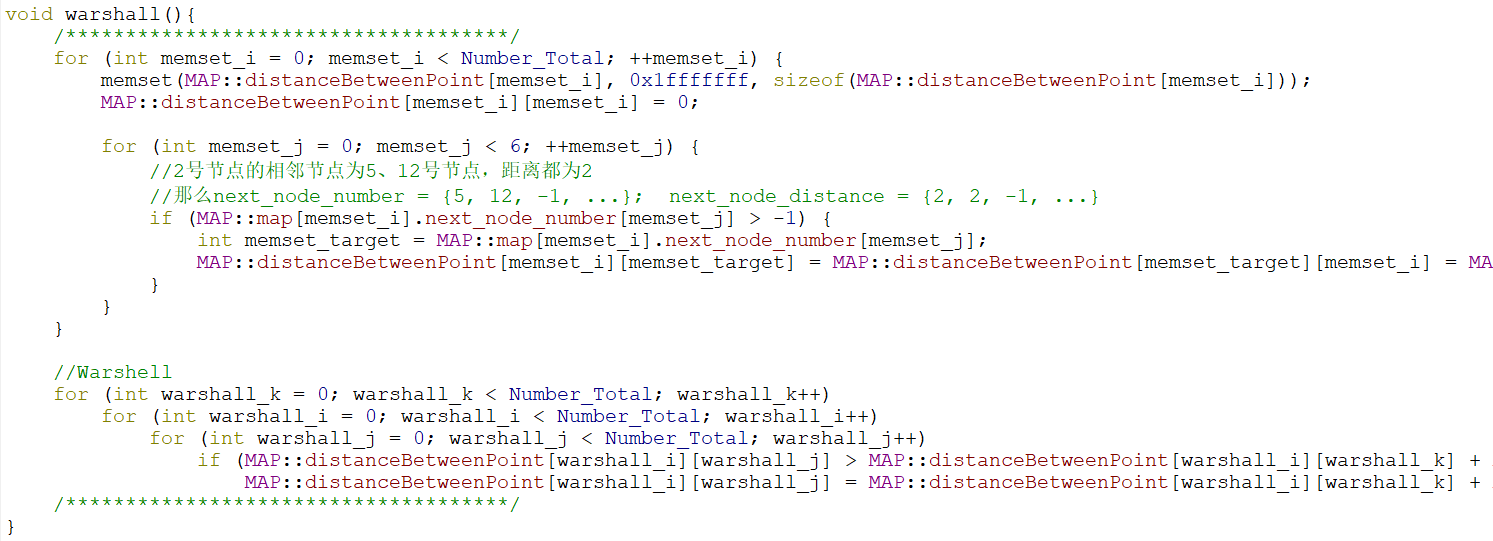


图2.6 warshell算法

## widget UI样式设置模块

本模块功能较为简单，主要是设置各类icon图标，背景图等。

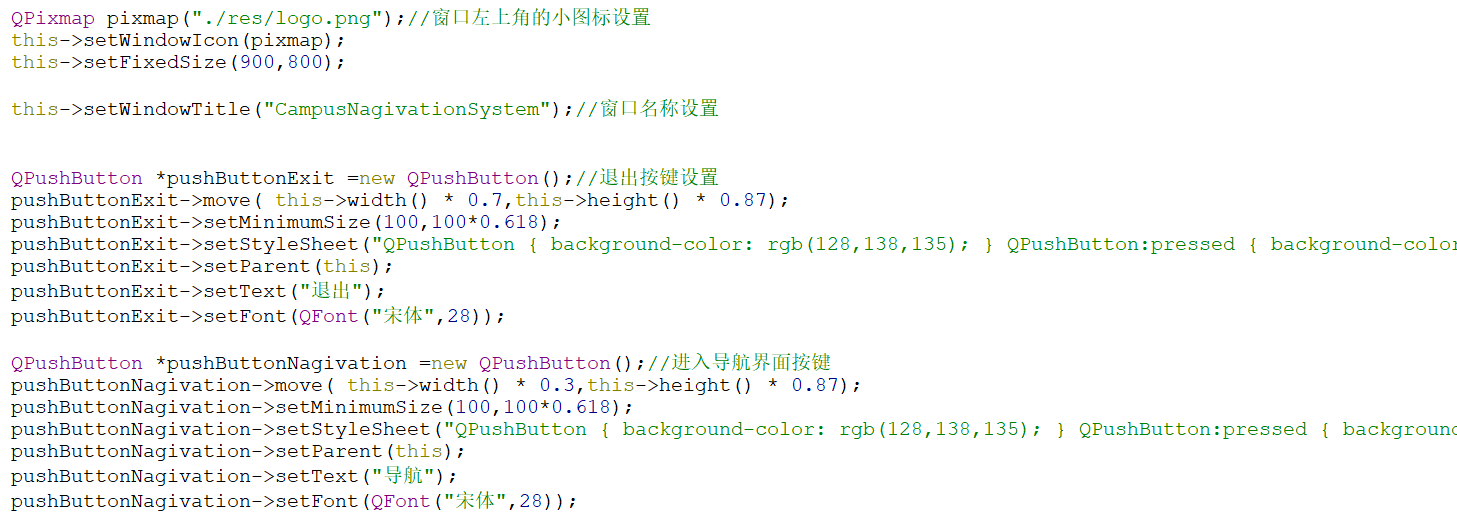


图3.1 UI样式设置

设置完成后，若用户选择进入程序，将显示主窗口。

## form 主窗口初始化模块

主窗口是用户进行操作和交互的界面，其中包含各类交换控件及按钮。界面交互部分如下图所示。

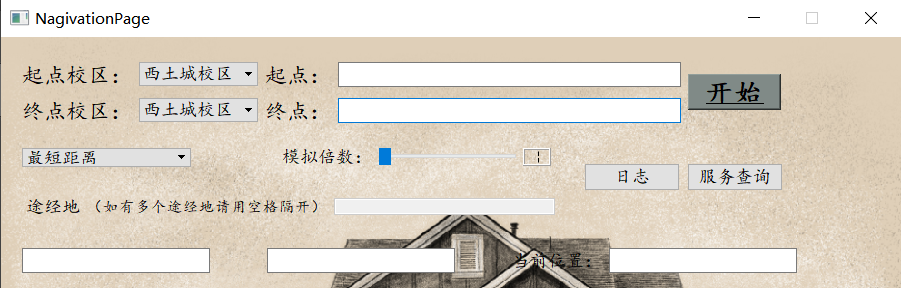


图4.1 主窗口交互部分

本模块的职能则是对这些窗口控件进行初始化，并绑定部分简单的控件槽函数。由于从窗口和日志窗口都是主窗口的成员类，所以它们的构造函数也会在此被调用。

由于本模块不存在算法相关内容，只是按部就班地顺序设定各控件参数，故不多赘述。后续的窗口初始化模块也一样。

## querywidget 从窗口初始化模块

从窗口主要用于向用户展示周边情况，包括周边的建筑以及服务等。日志窗口则展示用户的历史操作。日志显示分别为时间、起点代号、终点代号、导航策略代号。

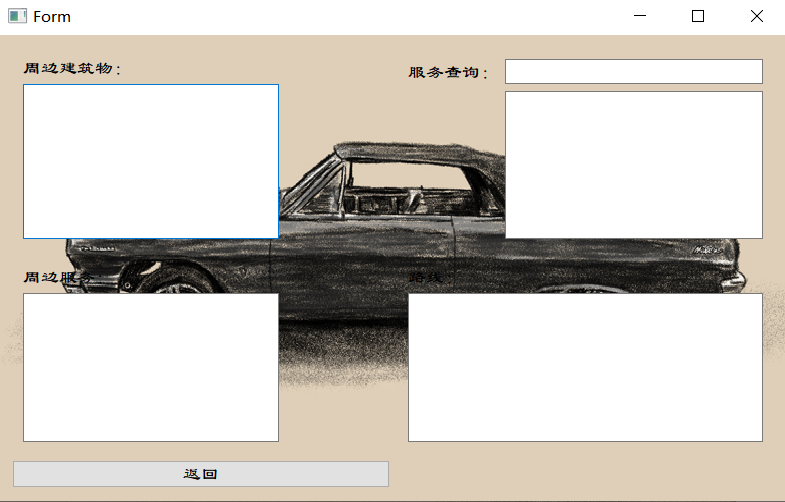


图5.1 从窗口界面

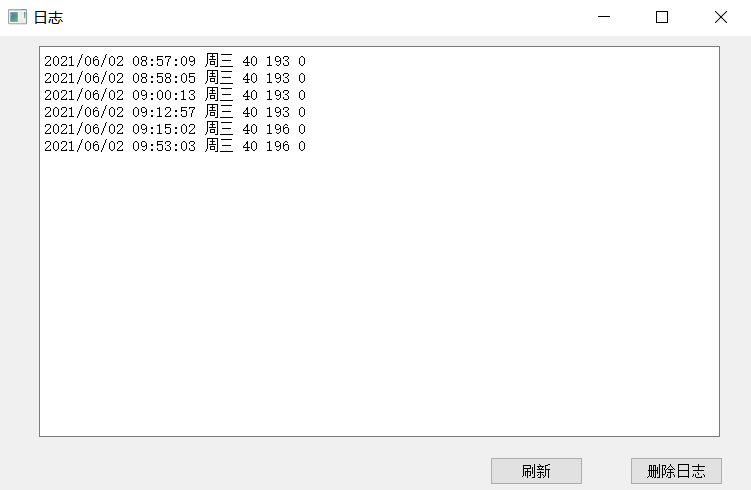


图5.2 日志界面

## buttonEvent点击事件处理模块

本模块将在用户点击“开始”时，从GUI界面获取用户的输入信息，并调用后端的规划路由模块，获取导航路线。而后还需根据路线中的道路状况，如拥挤度、校车/公交车等待之类的情况，计算出在每两个地图节点间的行进速度以及以该速度行进的距离。最终将这些信息返回给GUI界面，由GUI界面输出给用户。

除此之外，本模块还将根据导航策略限制用户的输入，例如在非途经点导航模式下，途经地输入框将不可用。若用户点击“日志”或“服务查询”按钮，本模块也将为其唤起相应的窗口。

模块流程图如下。

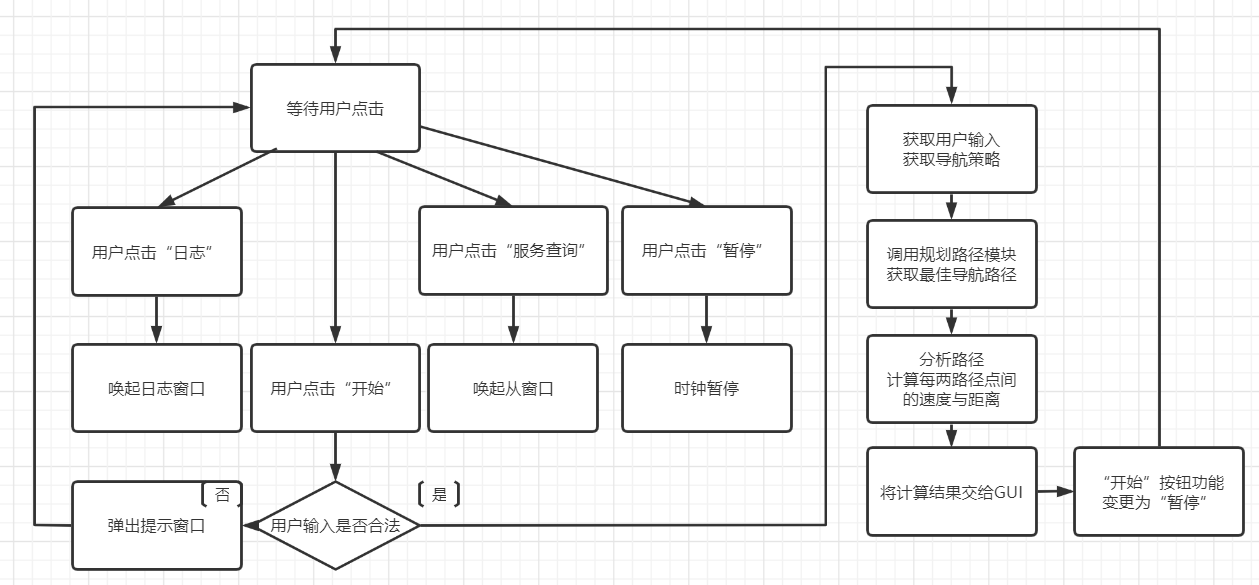


图6.1 模块流程图

主界面中还有“模拟倍数”控件，该控件的变化将导致时钟计时速度的变化，因此将其放在了aboutTime时钟模块中处理，本模块则不对该控件做操作。

本模块中涉及到与后端的对接以及对后端结果的二次处理，以下将介绍处理过程。

首先调用findTheWay函数，该函数是在findTheWay规划路径模块中定义的寻路函数，其返回值是一个int型的容器，容器中顺序存放了最佳路径所经过的各个地图节点。末尾再加上整条路径的实际距离和形式距离两个值。

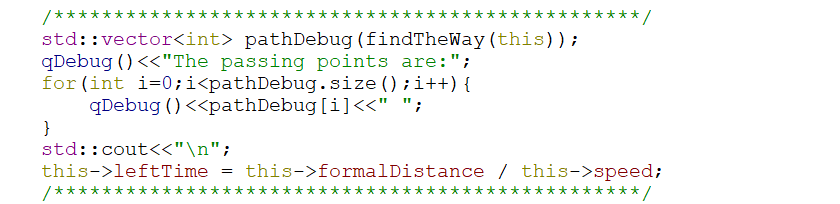


图6.2 调用规划路径模块获取最佳路径

形式距离，是根据道路拥挤度以及等车时间所等效出来的距离，拥挤的道路通行速度变慢，在时间上等效于速度不变的情况下走过更长的距离。等待时间同理，亦可以等效为一段距离。形式距离的作用是在“最短时间”策略中将拥挤度等时间因素或速度因素转换成距离因素，从而复用距离策略的算法。同时也能够方便地计算出本次导航所需的时间。

形式距离仅能用于后端计算路径时比较不同路径的优劣，而不能呈现给用户。因此在输出时，还必须将其转换回时间因素和速度因素。本模块的二次处理就是完成了这部分工作。

转换的思路是：计算每一条最小路段的实际通行速度，以及该段的距离，从第一个最小段开始，每该段的实际速度走过该段的距离后，将行进速度变更为下一个最小段的实际速度，再往前行进下一段距离。

对于每两个路径点间的道路，其实际通行速度等于默认速度除以1加百分之一的拥挤度。故可以使用循环算出每一段的实际速度，而后将该速度加入容器中，使其与该段的距离相对应。

除了距离之外，还需要对应记录路径点的代号，以便知晓当前时间点用户正处在哪个地方。

速度的变更将以剩余距离为触发条件。当剩余距离减小到一定程度时，即说明已经走过了一段路径。每一个触发速度变更的距离量称为变更距离。

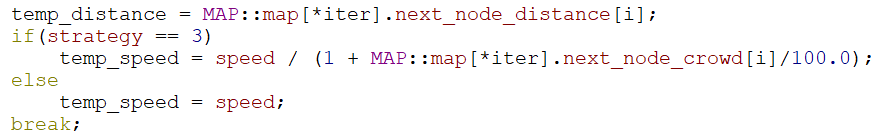


图6.3 计算每一个最小路段的实际通行速度

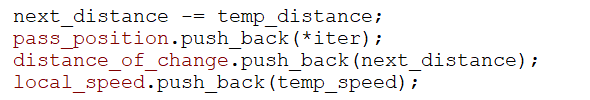


图6.4 记录变更距离、途经节点、通行速度

在跨校区导航中，用户需要等待校车/公交车的到来，而校车或公交车的默认速度与导航策略的默认速度不同，所以整体时间会出现偏差，这种偏差需要专门修正。

上文提过，本次导航所需的时间由形式距离算出，公式为：总时间 = 形式距离 / 默认速度。在地图中，两校区之间的距离很长，导致总形式距离很长，而默认速度是根据导航策略设定的固定速度，因此此时得出的总时间其实是以默认速度跨校区所需的时间。但实际上，校区间的校车速度远大于默认速度。

但是这个时间差是很好修正的，稍加思考即可知晓，时间差其实就是以默认速度走过校区距离和以校车速度走过校区距离之间的时间差。

而由于校区之间不存在拥挤度问题，所以等待时间就等于两校区间的形式距离与实际距离之差，再除以默认速度。

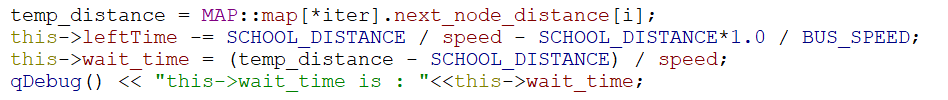


图6.5 计算等待时间、修正总时间

另一个跨校区导致的问题是，等待时用户速度为0，行进距离也为0，这就让原来的以剩余距离触发速度变化的方式无法使用了，也需要做出修正。

修正方法也并不困难，通过增加一个wait\_time变量来告诉时钟模块需要等待的时间，当时钟模块发现行进速度变为0时，就不再使用剩余距离来判定速度变化，而改用时间来判定。

如此一来，就实现了剩余距离和剩余时间相结合的判定方式，保证了跨校区时的正常显示。

## Recode 日志模块

日志模块以文件的方式记录用户的每一次导航，记录信息有四项，分别为导航时间、起点、终点、导航策略。

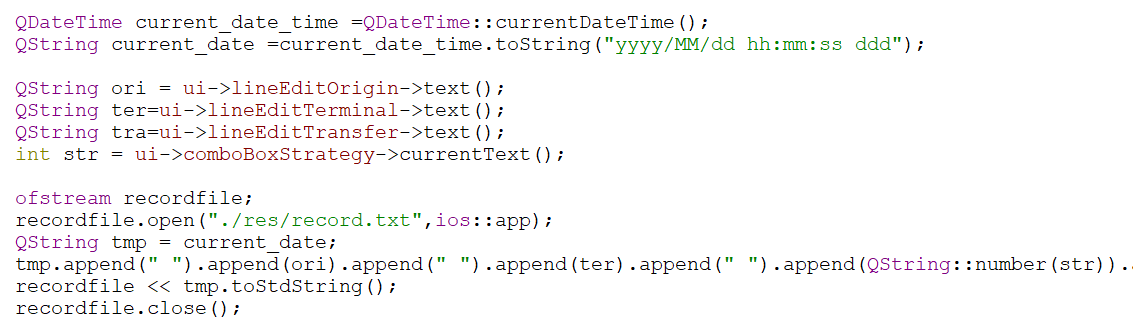


图7.1 以文件的方式记录日志

## selectCanteen 食堂均衡模块

程序运行过程中，将会记录综合食堂和教师食堂的人数，当用户选择的终点是人数较多的食堂时，将会弹出提示框，询问用户是否要前往人数较少的食堂。

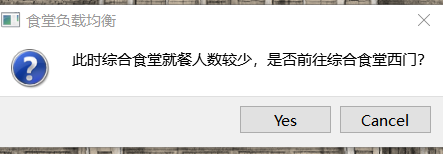


图8.1 食堂均衡负载

若用户选择是，程序将自动修改目的地，选择否则继续前往用户指定的目的地。无论用户选择哪里，程序将都会修改文件记录中的人数信息，达到动态调节的目的。



图8.2 食堂人数数据的读出和写回

## findTheWay 路由规划模块

程序中的寻路算法分为两层实现：路由规划和链路规划。两层次之间的关系类似于计算机网络中的网络层和数据链路层。

本模块承担“网络层”的功能。用户在导航过程中，可能需要途经多个地点，将这些地点看作是地图网络中的“路由器”，本模块负责找出穿过这些“路由器”的最佳顺序。

本模块实现排序的算法思路是“插缝”。

先考虑一种最简单的情况：只有一个途径点。此时，显然是“起点->途径点1->终点”的路径为最优。在此基础上，若再增加一个途径点，则有两种方案，一是“起点->途径点1->途径点2->终点”，二是“起点->途径点2->途径点1->终点”，这可以看作是将途径点2插入到起点和途径点1的缝隙之中，或插入到途径点2和终点的缝隙之中。

则显然，对于第n个途径点，它最多需要进行n次插入，就可以找到最适合它的那个位置。算法流程图如下：

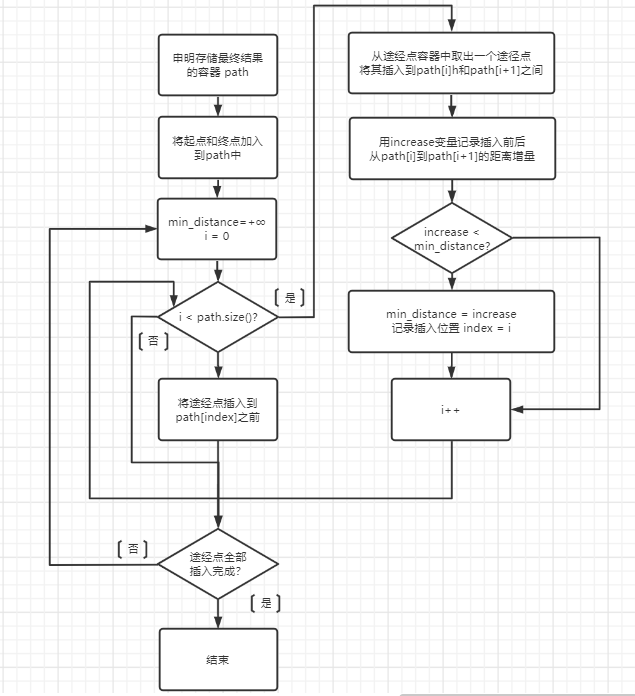


图9.1 “路由器”排序算法流程图

排序完成后，调用“链路层”的路线规划函数，找出每两个“路由器”之间的具体路径，而后将结果的距离累加，并赋值给Form类中的actualDistance成员。Form类是前端中的数据类，用于存放、接收后端的计算结果，并将其输出到GUI界面中。每次导航的具体路线也合并在一个容器中，返回给调用函数。

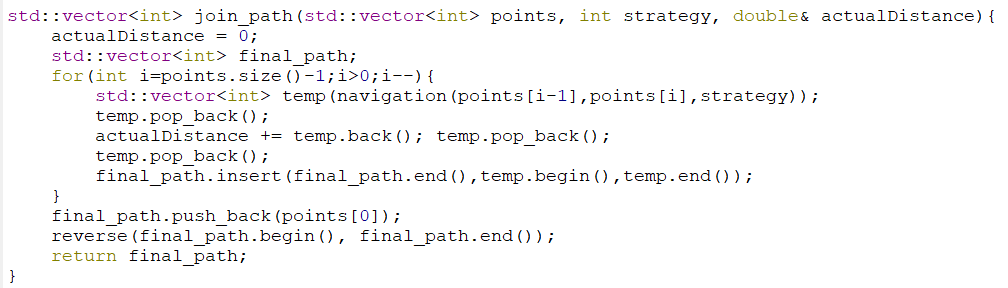


图9.2 路径拼接函数

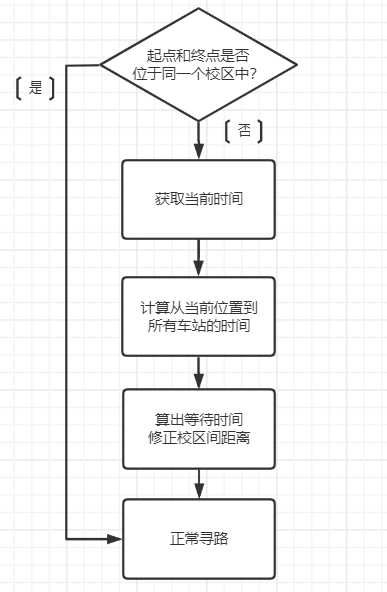
对于跨校区的导航，还需要插入关于校车或公交车的等待时间。我们假定公交车每10分钟有一班，而校车则根据时间表发车。

通过QTime类获取当前时间，将其完全转换成以秒计算的时间，例如，16点30分，转换为秒即是 （16\*60\*+30）\*60 = 59400，以此来标定一天内的时间。将获取到的当前时间存入current\_time中。

计算从当前位置到所有车站所需的时间，再加上current\_time，得到到达车站时的时间结点。公交车是每10分钟发车一次，所以用算得的时间结点除以600即可得知据上一次发车过去了多久，也就可以算出距离下一次发车还差多少时间。

而后将该时间转换成形式距离，加到该车站到另一校区全部车站之间的距离中去。

流程图如下：

  
图9.3 考虑公交车等待时间

对于校车，处理流程稍有差别。需要将校车发车时间表转换成秒计时，而后顺序储存在数组中，获取当前时间后，依次比对发车时间，等待时间等于第一个比当前时间大的发车时间与当前时间之间的差值。

若当前时间已经晚于最晚发车时间，则将等待时间设为无穷大。

修正完校区间的距离之后，即可正常调用下层函数，完成寻路。

## navigation 路线导航模块

本模块相当于“数据链路层”。负责找出任意两个地图节点之间的具体最佳路径，且支持多种寻路策略。

本模块中采用的是Dijkstra算法。经典的Dijkstra算法实际上是计算某一点到图中任意一点的最短距离的算法，其返回值仅为距离值，而没有具体路线。要使用该算法求出具体的行进路线，还需做一些小小的改变。

我们通过设立结果树来存储具体的路径。只需让算法迭代过程中找到的每一个点都记住它的上一个节点是谁(即记录父节点)即可，这样在计算完成时，从终点节点起不断的追溯其父节点，就可以一路追溯回起点。除了父节点之外，节点还需记录自身距离起点的实际距离和形式距离。实际距离是真正的物理上的距离，而形式距离是由于不同策略的影响，产生的仅供决策使用的等效距离。

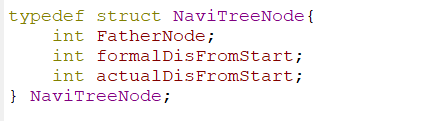


图10.1 结果树

对于策略区分问题，仔细分析后不难发现，不同的导航策略是可以使用同一套算法来解决的，只需要在无向图的边的权值(即节点间的距离)上做出一些修正，Dijkstra算法就依然适用。形式距离即是为此而生。

策略对于权值的影响实际上仅在于两个方面，一是以最短时间为目的时，需要考虑拥挤度，二是乘骑自行车时需要考虑道路能否通行。将拥挤度高的地区的形式距离加大，即可使算法将通过该区域的代价纳入考虑，而将不可通行的道路设为负数，即可使算法自然绕开该区域。

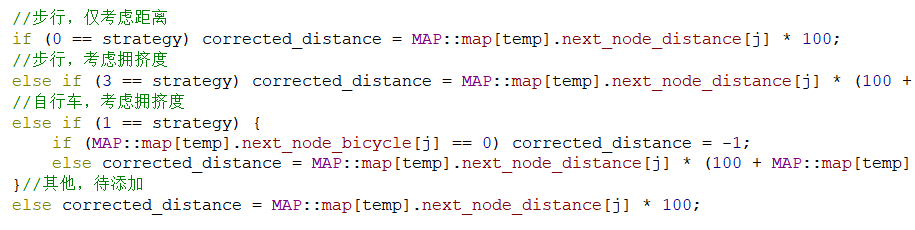


图10.2 策略修正

如此以来，本导航模块的难点迎刃而解了。Dijkstra算法的每一次迭代都会找出从起点到一个未标记点的最短路径，找出顺序按最短距离从小到大依次排列。因此当一个点被标记时，说明该点的最短路径已经得出，此时更新记录该点的两种距离，记录其父节点即可。找总体实现流程如下：

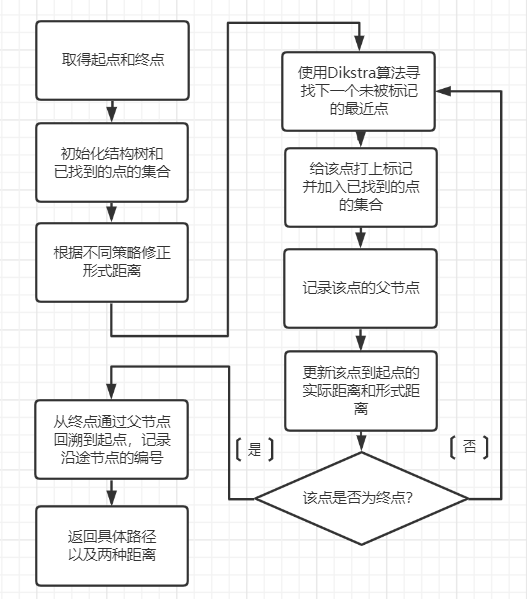


图10.3 模块整体流程图

## aboutTime 时钟模块

本模块的职能是为程序提供时钟计时，主要用于GUI显示时的剩余距离、剩余时间的刷新。时钟刷新周期为1毫秒。每次刷新时根据当前速度扣除对应的剩余距离。当剩余距离小于一定阈值时，说明走完了一段最小路段，此时变更速度。

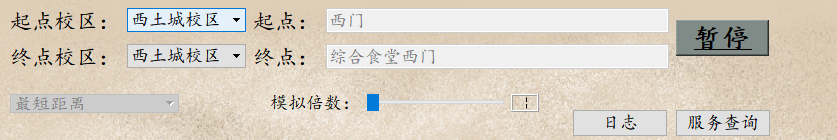
需要注意的是，等待校车/公交车时，速度为0，剩余距离不再变化，也不会触发新的速度变更。此时改用剩余时间作为触发速度变更的条件，等时间减少量等于类成员中的wait\_time变量的值时，速度变更为下一段路段的速度。

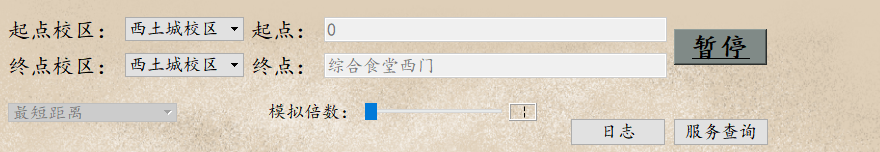
## IOfunction IO模块 / QIchange 字符串转换模块

这两个模块也是类似于分层调用的关系，它们共同完成对于用户输入的分析和转换。由于二者的实现都不涉及复杂算法，故一并描述。

用户的输入往往是直观的、具体的文字描述，而程序算法需要的是抽象的符号描述。将用户的具体描述转换成标准的、后端算法可以识别的编号就是本模块的职责。同时本模块也负责输出后端的计算结果，将其转换为标准名称。  
 转换的实现方式是建立映射表。建立文字描述与抽象编号间的多种映射关系，并将其存储在文件中，读取用户输入后，在映射文件中查找相对应的编号并返回。

以此为基础，程序支持多种输入方式。





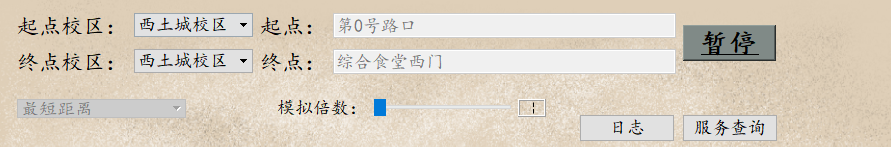


图12.1 支持多种输入方式

# 应用范例执行结果及测试情况说明

## 途径最短距离：

1. 程序开始时，导入地图数据。



1. 在输入框输入起点和终点，策略栏选择途径地最短距离，输入途径地，

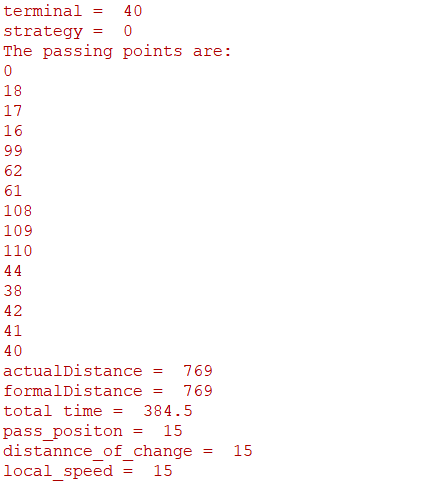
点击开始。







1. 数据经过正确转换后，传给后端处理。（下图分别为：终点、策略、路径、实际距离、形式距离、所需时间、经过的地点数、速度）



1. 实时（实际并非实时，而是每10毫秒更新一次状态，因为更新的足够快，可以认为是“实时”更新）查看到达目的地所需时间、与目的地的距离以及当前位置。



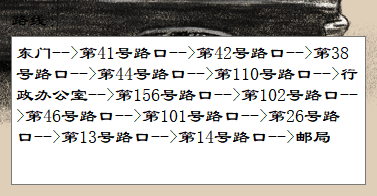




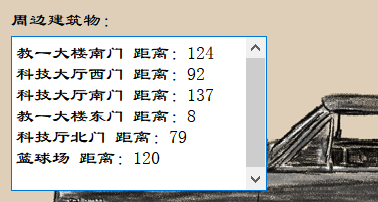


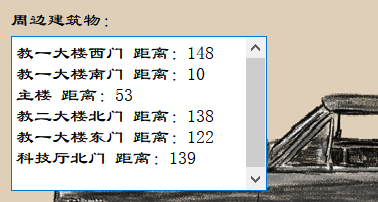


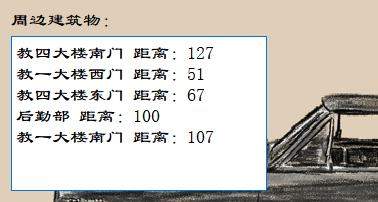
1. 路线显示。



1. 周边建筑物以及距离实时更新







1. 在地图中实时显示当前位置。

从起点出发：

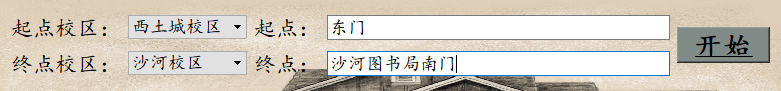
行走中：

到达（途径地）西门：

达到邮局：

## 跨校区导航：

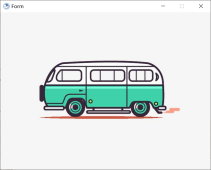
1. 选择校区，输入起点和终点，选择导航策略，点击开始。



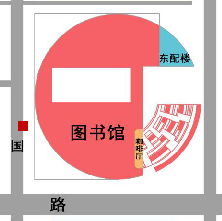


1. 从东门出发：

到达公交车站等待：

乘坐公交车中：

到达沙河西门：

到达图书馆南门：

# **七、评价和改进意见**

本程序较好地实现了实验要求，能够按照不同策略输出正确的路径，同时输出相应的时间日志以记录路径和时间。美中不足之处主要是程序的界面可以更为扁平化，从而进一步优化用户使用体验。

# 八、用户使用说明

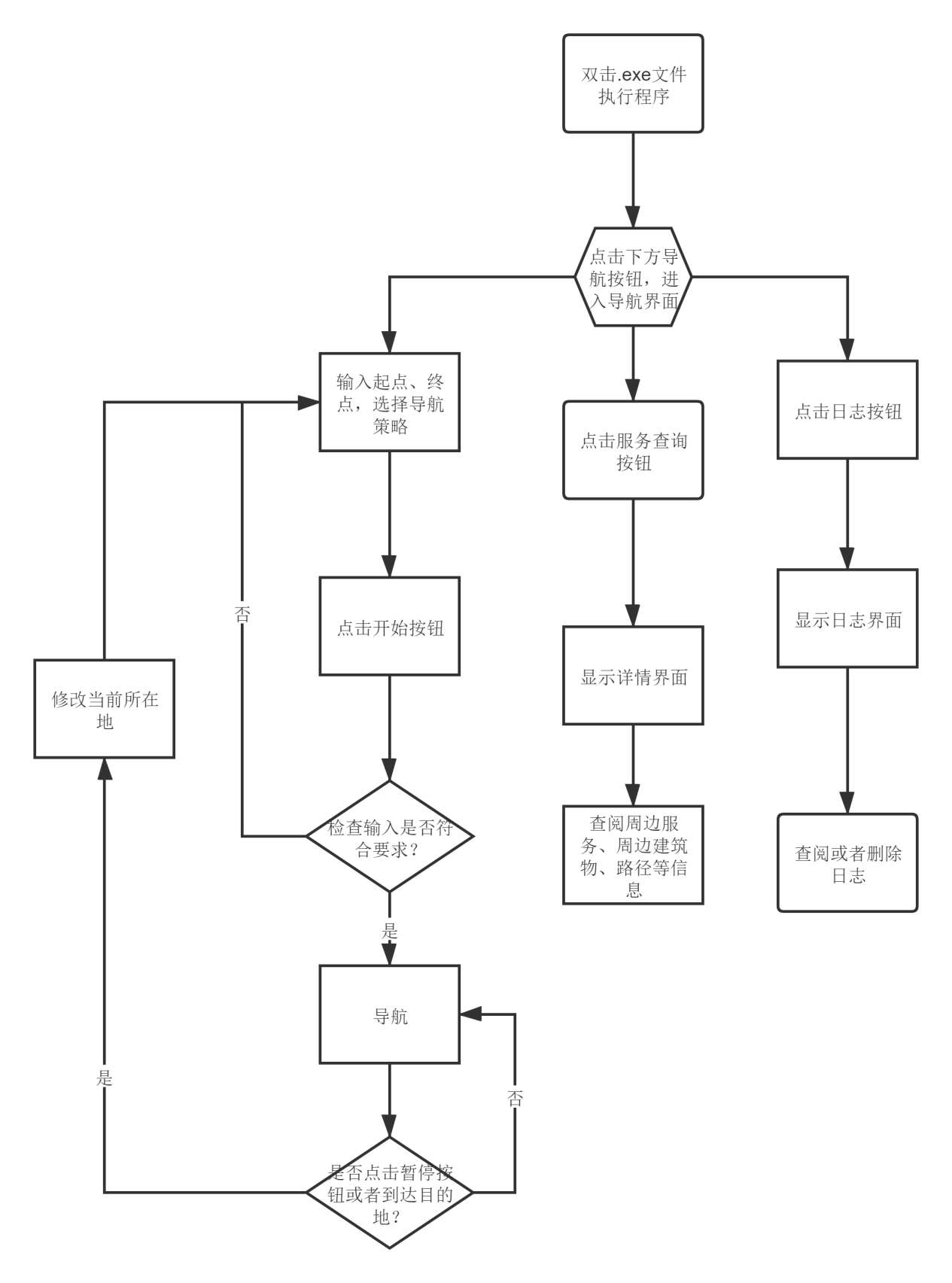


图4 导航软件使用指导图

此外，在图形界面开发中，我们额外编写了许多提示窗口，例如：“文件读取错误”、“地图载入失败”等程序错误提示和“目的地输入错误”、“确认是否暂停导航”等用户使用提示。简洁的用户界面以及丰富的提示信息大大地优化了用户的使用体验。