概要设计目录

一、高层数据结构设计

1.结构体定义

①地图的节点结构定义 node

```
typedef struct node{
    unsigned seq;   // 结点编号
    QString name;  // 结点名称
    QPoint COI;      // 中心点坐标
    QString campus;    // 结点所属校区
    unsigned symbol;  // 结点标记，如食堂、教学楼、公寓等
    unsigned people;  //  结点人数
}Node, *Np;
```

②路径中区段结构定义 section

```
typedef struct section{
    unsigned from, to;
    bool bicycle;              // 用户在当前区段骑自行车
    double curSpeed;        // 用户在区段[from,to]上的行进速度
    unsigned sectionCongestion;    // 当前区段的拥挤度
    unsigned remainingDistance;   // 距离下一节点的剩余距离
    unsigned sectionTime; //  到达下一节点的总时间
}Route, *Rp;
```

③用户结构定义 user

```
typedef struct user{
    QString userName, userState; // 用户名、用户状态
    QString strategy, campus;      // 行进策略、用户当前所处的校区
    QString start_time, end_time;// 记录一次导航的开始与结束时间，主要用于日志记录

    Node src, dst, transPoint;       // 起点、终点、中转点
    Node srcAccurate, dstAccurate, transAccurate; // 可能存在的精确制导点
    unsigned timecost, walked;   // 走过的总时间和距离
    QPoint location;      // 用户的当前位置

    QList<Rp>   Path;  // 总的行进路线
    Rp curNode;         // 行进路线中的当前节点
}User, *Up;
```

④班车记录结构 busRecord

```
typedef struct busRecord{
    QString date;  // 日期
    QString origin, destination; // 出发地、目的地
    QString start_time, arrive_time; // 开始时间、结束时间
    QString type;  // 交通方式，"校车"、"班车"
}bR, *bRP;
```

2.全局变量及宏定义

变量声明	解释说明
#define MAX_NUM 256	地图中最大节点数
#define INF 0x3f3f3f3f	不可达距离
#define coordinateCorrect QPoint (20,25)	模拟导航时需要排除用户标签自身的大小
#define Search_Ridus 500	查询功能中的默认查询半径(m)
#define TIME_FLAG_TRUE 1	等待状态下的时间标志，系统时间 1s=真实时间 1s，防止用户设置参数时，时间流逝过快。
#define TIME_FLAG_CAMPUS 10	校区内移动时的时间标志，系统时间 1s=真实时间 10s。
#define TIME_FLAG_WAITBUS 60	等车状态下的时间标志，系统时间 1s=真实时间 1min。
#define TIME_FLAG_CITY 300	校区间移动时的时间标志，系统时间 1s=真实时间 5min。
int walkSpeed=1, bicycleSpeed=3, busSpeed=1;	设置用户步行、骑行以及公交车的理想速度。
int Graph[MAX_NUM][MAX_NUM];	节点连通矩阵，其值为两节点间的距离。
double Congestion[MAX_NUM][MAX_NUM];	拥挤度矩阵，其值为两节点间的拥挤度。
bool Bicycle[MAX_NUM][MAX_NUM];	骑行矩阵，其值为两节点间是否可以骑行。
QMap<unsigned, Np> graph_nodes;	地图节点映射，将节点序号与节点结构关联。
QVector<bRP> busTable;	交通时刻表
User uuser;	用于模拟导航的全局用户

3.导航主界面中的私有变量

变量声明	解释说明
QLabel *permanent;	置于主界面右下状态栏中显示时间的标签
QDateTime datetime;	系统时间对象
QTimer *sysTimer;	用于更新系统时间的定时器
QTextToSpeech *tts;	语音播报对象
bool ttsFlag=false;	标志当前是否启用了语音播报功能
bool accurateFlag=false;	标志当前是否启用了精确导航功能
QString msgCache;	导航信息缓存
bool movePause=false, pauseFlag;	前者记录当前是否处于暂停状态，后者记录上一区段路程是否暂停过
Dialog *load;	校区转换时的中转加载框
QList<QString> commonPath;	存储路径相关的偏好设置信息
QStringListModel pathModel;	将路径偏好用视图展示出来
QStandardItemModel *_matchModel, *_searchModel;	前者用于将编辑时按关键字匹配的搜索结果展示出来，后者用于展示周边建筑物查询结果。
QString whichMap;	记录当前使用的导航平台图的名字
QTimer *userTimer;	用于模拟移动过程中记录用户移动的区段计时器
QTimer *moveTimer;	用于控制用户移动的单位计时器
QTimer *randTimer;	用于更新建筑物内人数的随机数计时器

二、系统模块划分

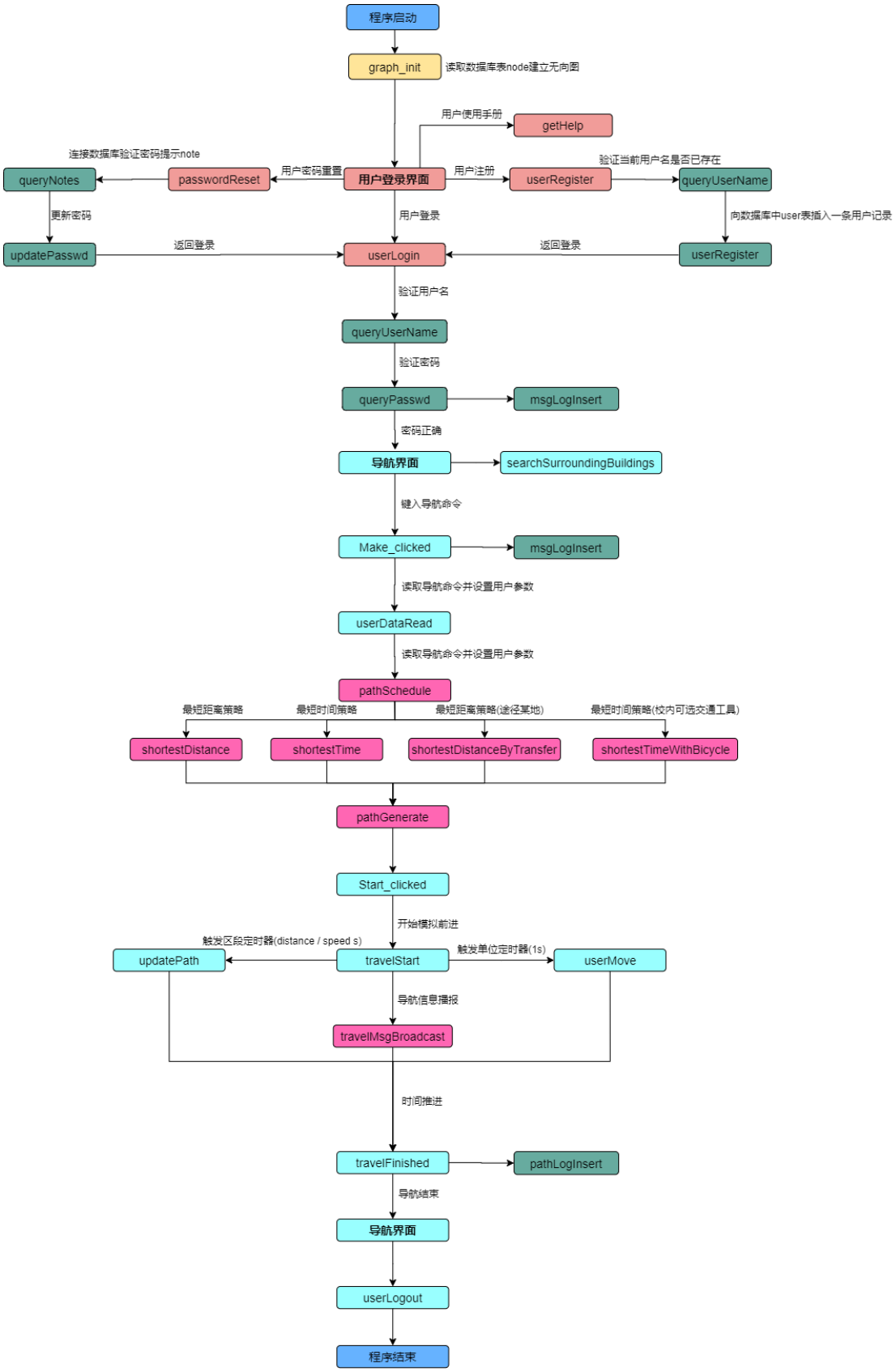
1.系统模块划分思路

整个程序大体上可分为数据库连接 database、用户登录注册的管理 manage、常用函数设计 public、主体算法设计 guide、导航界面显示及界面响应 campusGuide 五个模块。

模块名称	主要功能
database	连接数据库；定义函数接口用于用户信息、键入命令、路径信息等数据的存储。
public	读取数据库中存储的静态数据建立无向图及邻接矩阵；提供一些常用函数，如用于两点间计算距离的函数。
manage	负责用户注册、登录、提供用户使用手册。
guide	路线规划算法；扩展功能实现。
campusGuide	加载地图，读取用户键入信息，调用功能模块的路线设计函数，并根据路线模拟前进。

2.模块关系图及程序流程图

模块调用几乎完全串行，直接给出程序流程图，不再赘述函数调用关系。



3.各模块函数接口说明

函数原型	功能说明
void graph_init()	从数据库表 node 中读取结点数据建立无向图并初始化邻接矩阵。
int distance(unsigned A, unsigned B)	返回 A→B 的几何距离。
int distance(QPoint &A, QPoint &B)	返回 A→B 的几何距离。
void busTable_init()	交通时刻表初始化。
void specialNodeAvaliable()	室内导航时，将室内外连接处的特殊节点连通。
void specialNodeForbidden()	室外导航时，禁用室内外连接的特殊连接节点，防止导航到室内。
unsigned getIndexOfGraph(QString nodeName, QString campus);	根据节点名、所属校区返回其对应于图的序号。
unsigned getIndexOfGraph(QString nodeName);	当寻找精确导航的室内节点时，节点名唯一，可直接查找。
void myMessageOutput(QtMsgType type, const QMessageLogContext &context, const QString &msg)	基于系统特性存储的程序运行日志。
database *database::getDatabase()	创建一个数据库实例
bool database::userRegister(const QString &userName, const QString &passwd, const QString &notes)	用户注册
bool database::queryNotes(const QString &userName, const QString &notes)	重置密码时验证关键字
bool database::queryUserName(const QString &userName)	登录、注册时检测用户名是否已存在
bool database::queryPasswd(const QString &userName, const QString &passwd)	登录时的密码检测
void database::updatePasswd(const QString &userName, const QString &passwd, const QString &notes)	重置密码时的更新操作
static void database::getPreferredPath(const QString userName, QList<QString> &commonPath)	【静态函数】获取日志记录中用户常用的路径
static bool database::msgLogInsert(const QString logTime, const QString userName, const QString userState, const QString logMsg)	【静态函数】向数据库中插入一条用户操作记录。
static bool database::pathLogInsert(const QString userName, const QString timeStart, const QString from, const QString trans,	【静态函数】当导航结束时，向数据库中插入一条路径日志记录。

const QString to, const QString strategy, const QString pathMsg, const QString timecost)	
void manage::uiSetLoad()	加载登录界面的一些 ui 设置
void manage::switchPage()	页面切换槽
void manage::loadCfg()	加载配置文件并读入信息，如果自动登录有效则发出 autoLogin()信号
void manage::saveCfg()	保存配置文件信息
void manage::on_pushButton_help_clicked()	用户手册按钮的响应槽
void manage::on_autoLogin_stateChanged(int arg)	当自动登录状态改变时修改配置文件
void manage::on_remPasswd_stateChanged(int arg)	当记住密码状态改变时修改配置文件
void manage::on_pushButton_login_clicked()	登录按钮的响应槽
void campusGuide::objcetAndSignalsInit()	【初始化】对象实例化并关联信号槽
void campusGuide::uiSetLoad()	【初始化】加载界面的初始设置
void campusGuide::buttonStateChange(int arg)	【按钮使能控制】根据参数控制各个状态下的按钮使能
void campusGuide::userLabelChange()	【行进时】根据行进方向及交通方式修改用户标签的显示图片
void campusGuide::userDataRead()	【制定路线时】获取当前行编辑器内的参数设置导航信息
void campusGuide::travelStart()	【模拟行进】行进调度
void campusGuide::travelFinished()	【模拟行进】行进结束
void campusGuide::updateTime()	【模拟行进】系统时间标签的时间更新槽
void campusGuide::updatePath()	【模拟行进】每个路径区段完成时，更新当前路径参数及地图切换等
void userMove()	【模拟行进】每触发定时器，用户移动一次
void mapChange()	【模拟行进】当导航的校区改变时切换显示的地图
void updatePeople()	【偏好设置】每分钟更新一次建筑物内的人数
void commandParse()	【命令分析】命令分析调度函数
void preferredPathShow()	【偏好设置】获取并展示当前用户的常用路径
static void pathSchedule()	【路径调度】根据用户参数进行路径规划
static void travelMsgConcat(QString &msg)	【路径信息连接】导航信息实时播报并将信息返回做记录
static QString timeTransfer(unsigned time)	将毫秒时间转化为(hh:mm:ss)形式的时间串

static void **shortestDistance**(QList<unsigned> &myPath, unsigned from, unsigned to)	【路线规划】最短距离策略下的路线规划函数
static void **shortestTime**(QList<unsigned> &myPath, unsigned from, unsigned to)	【路线规划】最短时间策略下的路线规划函数
static void **shortestDistanceByTransfer**(QList<unsigned> &myPath, unsigned from, unsigned to)	【路线规划】最短距离（途径某地）策略下的路线规划函数
static void **shortestTimeWithBicycle**(QList<unsigned> &myPath, unsigned from, unsigned to)	【路线规划】最短时间（校内可选交通工具）策略下的路线规划函数
static void **pathGenerate**(int path[][MAX_NUM], QList<unsigned> &myPath, unsigned u, unsigned v)	【路径输出】根据路线规划函数得到的路径矩阵 path 输出 u→v 的路径
static void **makePath**(QList<unsigned> &myList)	【路径输出】根据得到的路径进一步制定可用于导航的精确路径。
static QString **pathOutput**()	【路径输出】输出当前用户的路线
static bool **isCampusMoving**()	判断当前区段是否是校区间中转
static void **pathLogInsert**(QString msg)	【日志记录】导航结束后插入一条路径记录日志
static void **msgLogInsert**(QString msg)	【日志记录】插入一条操作记录日志
static void **searchSurroundingBuildings**(QString name, unsigned ridus, QList<QString> &_strList);	【查询】查询用户当前位置周边的建筑物，辐射半径为 ridus

三、高层算法设计

1.路线规划算法的设计及最优性证明

仔细分析四种策略下的路线规划算法容易发现，本质上它们都是带权无向图的多源最短路径问题，不同的策略差异只表现在无向图的权值上。因此，它们可以使用一个核心的路线规划算法来解决。在综合考量 dijkstra 和 floyd 算法的实用性和可推广性，最后决定采用 floyd 算法来实现路线规划。

下面给出最短距离策略的 floyd 算法作示例：

```
void campusGuide::shortestDistance()
{
    int n=graph_nodes.size(); // 顶点个数
    int path[MAX_NUM][MAX_NUM];  // path[i][j]记录从i→j经过了哪个点
    memset(path, -1, sizeof path);

    int graphTemp[n+1][n+1];
    for(int i=1;i<=n;i++)
        for(int j=1; j<=n;j++)
            graphTemp[i][j] = Graph[i][j];

    // Floyd算法
    for(int k=1;k<=n;k++){ //选中的中间值
        for(int i=1;i<=n;i++){ //数组横坐标
            for(int j=1;j<=n;j++){ //数组纵坐标
                if(graphTemp[i][j]>graphTemp[i][k]+graphTemp[k][j]){     //如果以k中间点为中间点检测到路径更短
                   graphTemp[i][j]=graphTemp[i][k]+graphTemp[k][j];       //更新路径值
                   path[i][j]=k;    //更新要经过的中间点
                }
            }// end of for j
        }// end of for i
    }//end of for k


    qDebug() <<"路径长度: " << graphTemp[uuser.src.seq][uuser.dst.seq];
    QList<unsigned> myPath;

    pathGenerate(path, myPath, uuser.src.seq, uuser.dst.seq); // 根据得到的floyd结果矩阵输出节点序列
    myPath.append(uuser.src.seq);
    qDebug() <<uuser.src.seq;
    makePath(myPath);   // 由输出的路线指定用于导航的路径path
}
```

下面简单证明一下途径中转点的最短距离问题是可以通过起点到中转点，再由中转点到终点的两次 floyd 算法得到解决的。首先需要明确的是，不同于 dijkstra 算法是基于贪心思想，floyd 算法是动态规划的。而要证明 floyd 中转点问题可以由两次 floyd 算法解决，传统意义上应该证明该问题同时具备最优子结构性质与子问题重叠性质。但我们注意到，floyd 算法的本质就是通过一个中转点对节点间的连通性作传递，因此算法本身就是一个必经中转点的算法，只是这个中转点从 1~N 被遍历了 N 次。

我们这里从有 N 个节点的无向图入手进行证明。若图中两个节点不连通，那么经过算法计算后仍为不连通。若它们连通就必然存在一条最短路径。假设任意 2 节点之间存在一条最短路径，该路径上有 M 个节点（M<N）。floyd 算法为三重循环，我们重点考虑最外层循环，我们将最外层循环此时所在的节点称为“桥点”。桥点遍历到了某一节点时，算法就会计算任意一点 A 到桥点的距离加上任意一点 B 到桥点距离的和，并将这个和与矩阵中存储的 A 到 B 的路径的距离进行比较，将短的存入矩阵中相应的位置。

2.模拟用户移动的实现

模拟用户移动主要通过定时器实现，每触发一次单位定时器，就根据流逝的时间乘以速度将用户作移动。而每触发一次区段定时器，就代表当前区段已经完成，切换到下一个区段继续移动，直到终点。

其中最主要的难点不在于定时器的设置，而是如何让程序在定时器计时器间等待同时保持主窗口响应，最终采用 Qt 的局部事件循环 QEventLoop 让程序在该时间内局部休眠，成功模拟了用户移动过程。

主体代码如下：

```
// 跟随制定的路线的模拟行进
for(int i=0; i < uuser.Path.size(); i++)
{
    auto curNode = uuser.Path[i];
    uuser.location = QPoint(graph_nodes[curNode->from]->COI);

    if(uuser.strategy == 3 && Bicycle[curNode->from][curNode->to])     // 当前为可选交通工具的行进策略
        curNode->curSpeed = bicycleSpeed;
    else
    {   // 其他策略仅考虑拥挤度的影响
        double cg = curNode->sectionCongestion;
        cg = 1 - cg/10;
        curNode->curSpeed = walkSpeed * cg;    //当前路段的真实速度
    }

    ui->userLabel->move(graph_nodes[curNode->from]->COI);
    curNode->remainingDistance = Graph[curNode->from][curNode->to];
    curNode->sectionTime = curNode->remainingDistance / curNode->curSpeed;
    uuser.curNode  = curNode;
    uuser.walked += curNode->remainingDistance;

    QString msg = QString("[%1]").arg(i+1);
    travelMsgBroadcast(msg);
    msgCache = msgCache + msg + '\n';
    ui->textBrowser_msg->setText(msgCache);
    ui->textBrowser_msg->repaint();

    // 开启定时器,模拟行进过程及时间推进
    connect(userTimer, SIGNAL(timeout()), this, SLOT(updatePath()));
    userTimer->start(1000/timeFlag * curNode->sectionTime); // 系统时间为6倍真实时间推进
    moveTimer->start(1000);

    // do nothing but wait
    QEventLoop eventLoop;   // 使用局部事件循环让主线程等待，但仍保持响应。
    QTimer::singleShot(1000/timeFlag * curNode->sectionTime, &eventLoop, SLOT(quit()));
    eventLoop.exec();

    uuser.timecost += curNode->sectionTime;
}//end of for
ui->userLabel->move(uuser.dst.COI);
ui->userLabel->repaint();
```