****

**XXXXXXX课程设计**

**实验报告**

|  |  |
| --- | --- |
| 学生姓名 |  |
| 学 号 |  |
| 专业班级 |  |
| 指导教师 |  |
| 学 院 | 计算机学院 |
| 完成时间 | 2022.1.11 |

计算机学院

目录

[1.系统描述 4](#_Toc93139046)

[2.功能模块结构 5](#_Toc93139047)

[制作地图界面功能 5](#_Toc93139048)

[查询系统界面功能 7](#_Toc93139049)

[3.主要模块算法说明 9](#_Toc93139050)

[1.绘图函数 9](#_Toc93139051)

[2.编辑按键和功能号和鼠标响应事件 9](#_Toc93139052)

[3.文件的保存和读取 11](#_Toc93139053)

[4.零碎的基础函数 11](#_Toc93139054)

[5.dijkstra和prim最小生成树算法 13](#_Toc93139055)

[6.回溯法求所有路径 15](#_Toc93139056)

[4.运行结果 16](#_Toc93139057)

[主界面： 16](#_Toc93139058)

[地图制作界面 16](#_Toc93139059)

[地图查询界面： 20](#_Toc93139060)

[5.课程设计总结 22](#_Toc93139061)

[附录 23](#_Toc93139062)

[Untitled1.pro 23](#_Toc93139063)

[Creat\_map.h 24](#_Toc93139064)

[Mainwindow.h 26](#_Toc93139065)

[Map\_query 26](#_Toc93139066)

[Showmess.h 27](#_Toc93139067)

[Creat\_map.cpp 28](#_Toc93139068)

[Main.cpp 48](#_Toc93139069)

[Mainwindow.cpp 48](#_Toc93139070)

[Map\_query.cpp 49](#_Toc93139071)

[Showmess.cpp 61](#_Toc93139072)

[Creat\_map.ui 61](#_Toc93139073)

[Mainwindow.ui 68](#_Toc93139074)

[Map\_query.ui 70](#_Toc93139075)

[Show\_mess.ui 75](#_Toc93139076)

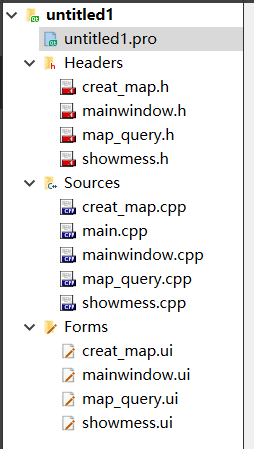
# 1.系统描述

搭建了一个校园景点规划和导游的系统，设计学校的校园平面图。系统分为两个入口，一个是设计地图界面，一个是查询地图界面。

设计人员可以在设计地图界面处制作地图，导入一张地图图片后利用系统功能实现添加点和边，删除点和边，修改点和边的信息，然后把地图保存为特定格式的文件，系统还提供读取这类特殊文件的功能。

访问人员可以在查询地图界面查找两点间最短路径，两点间所有路径，图的最小生成树，查询景点信息，但无法修改地图。





项目结构如右图所示：

其中creat\_map为设计地图界面

map\_query为查询地图界面

showmess为显示地点信息界面

mainwindow为初始界面，即主窗口界面

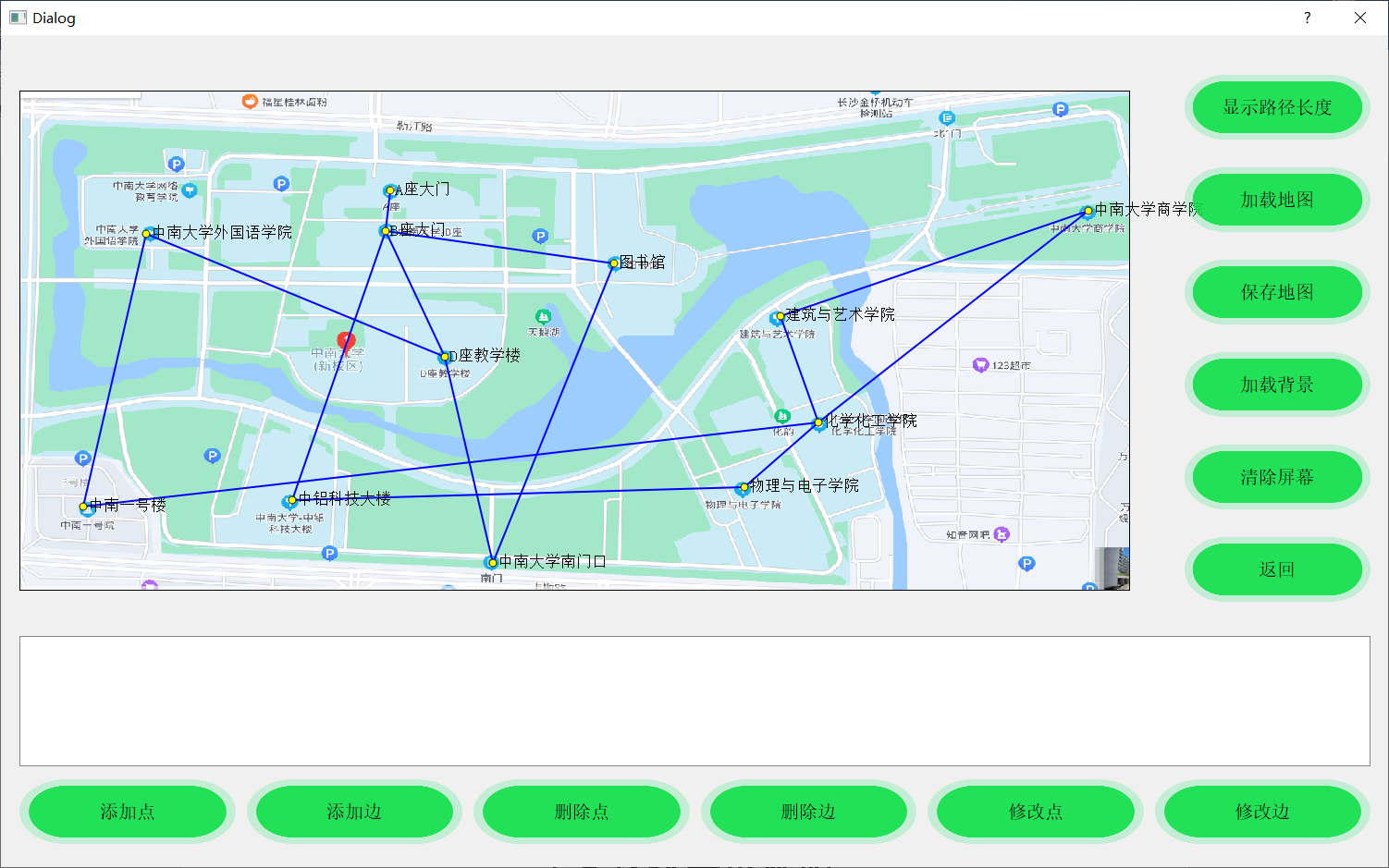
main为主函数。

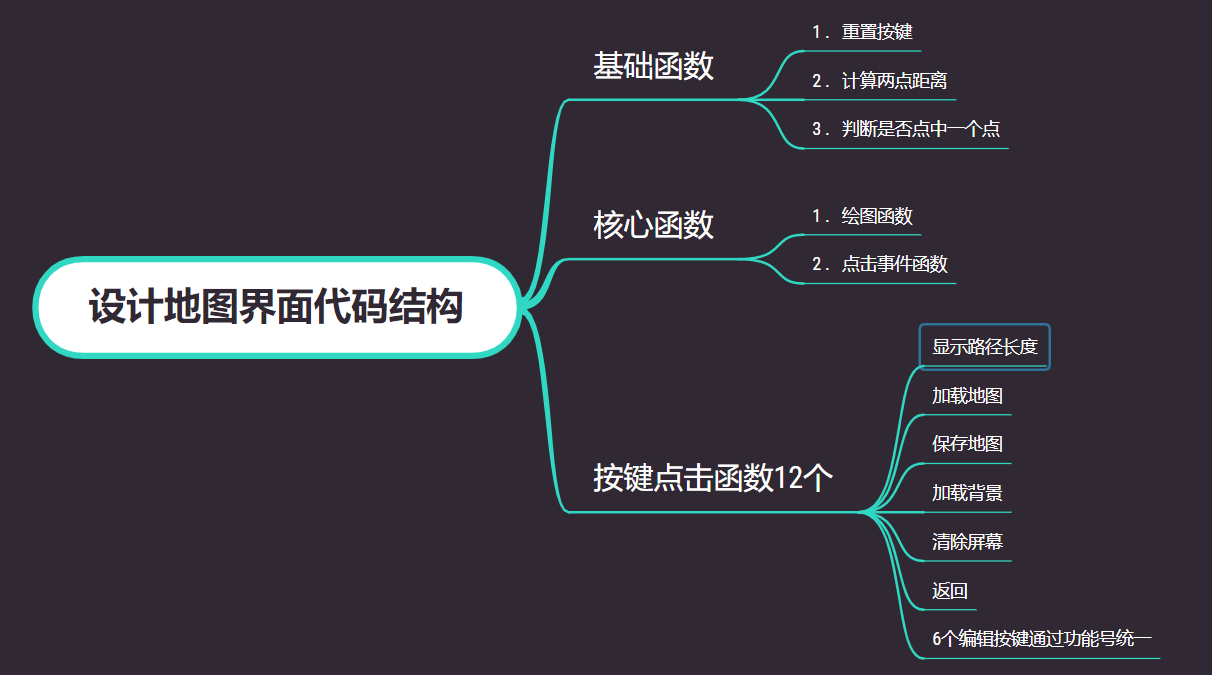
# 2.功能模块结构

## 制作地图界面功能

显示边的权重，加载地图，保存地图，加载背景，清除背景

加点，加边，删点，删边，改点，改边





**基础函数：**

void **Recover**();

按钮信息重置

double **Count\_distanse**(QPoint p\_1, QPoint p\_2);

计算两点间距离

bool **is\_Click\_Suc**(QPoint x, QPoint y, int n);

判断点x,y是否相当于同一个点，即鼠标是否点击到了某个点

**核心函数：**

void ***paintEvent***(QPaintEvent\*);

绘图函数

void ***mousePressEvent***(QMouseEvent\* event);

鼠标点击事件

**按键点击函数12个**

void **on\_back\_clicked**(); //返回按键

void **on\_del\_node\_clicked**(); //删除点

void **on\_add\_node\_clicked**(); //添加点

void **on\_del\_edge\_clicked**(); //删除边

void **on\_add\_edge\_clicked**(); //添加边

void **on\_mod\_node\_clicked**(); //修改点

void **on\_mod\_edge\_clicked**(); //修改边

void **on\_showlen\_clicked**(); //展示路径

void **on\_load\_map\_clicked**(); //加载地图

void **on\_save\_map\_clicked**(); //保存地图

void **on\_load\_bg\_clicked**(); //加载背景

void **on\_clear\_bg\_clicked**(); //清除背景

## 查询系统界面功能

找两点间最短路径，找两点间所有路径，找最小生成树，查询信息





**基础函数：**

void **Recover**();

按钮信息重置

double **Count\_distanse**(QPoint p\_1, QPoint p\_2);

计算两点间距离

bool **is\_Click\_Suc**(QPoint x, QPoint y, int n);

判断点x,y是否相当于同一个点，即鼠标是否点击到了某个点

void **drawArrow**(QPoint startPoint, QPoint endPoint, QPainter &p)

画带箭头的线

void map\_query::***paintEvent***(QPaintEvent\*)

绘图

void map\_query::***mousePressEvent***(QMouseEvent\* e)

鼠标点击事件

**核心函数：**

bool **dijkstra**(int a, int b)

最短路径

void **dfs**(QVector<QVector<int>> &res, QVector<bool>& flag, QVector<int>& temp, int cur, int end)

void **allPathsSourceTarget**(int a, int b)

所有路径

QVector<QPair<int,int>> **prim**()

prim最小生成树

**按键功能函数六个：**

void **on\_back\_clicked**(); //返回按键

void **on\_load\_map\_clicked**(); //加载地图

void **on\_shortest\_rode\_clicked**(); //最短路径

void **on\_all\_rode\_clicked**(); //所有路径

void **on\_mintree\_clicked**(); //最小生成树

void **on\_clear\_clicked**(); //清除背景

# 3.主要模块算法说明

## 1.绘图函数

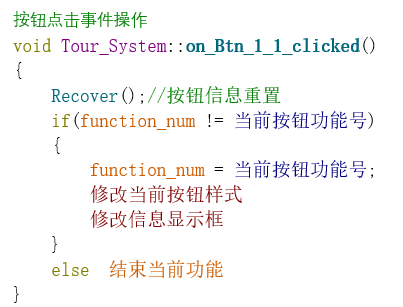
绘图这一块主要是对QT自带的各种绘图函数的应用。先建立一个QPainter对象painter，再建立两种QFont的字体对象font1，font2用来绘制界面不同部分。先用setMaximum，SizesetMinimumSize函数将应用程序的界面限制在固定的大小防止用户放大缩小后导致界面变形。painter.drawPixmap画出大背景，painter.drawRect画矩形框，painter.setFont设置字体样式，painter.setPen设置画笔参数，painter.drawLine由两点画出一条线，painter.drawText在图中画出文字，painter.drawEllipse画出椭圆（这里用来画圆），painter.setBrush填充颜色。

画图时的思路：先画出线再画点，防止点被线覆盖住。对于地图查询界面，遍历每条线，如果线的flag属性为1，则这条线是最小生成树，用红线画。如果flag属性为0，则这条线是普通的线，用蓝线画（点击最小生成树功能时会将数中的线的flag置1，取消后会将所有线的flag复原为0）。然后检查show\_len\_flag标志判断是否要标出线的权重，如果要则遍历每条边在边的中间标出边的权重，最后遍历所有的点，取出点的x,y坐标用painter.drawEllipse画出点。

## 2.编辑按键和功能号和鼠标响应事件



为了编写时的方便和清晰，对点和边进行加，减，改操作的六个功能按键进行了统一，并另设了一个全局变量功能号function\_num。即编辑操作的六个按键都归一化为一种格式如下：



**例如添加点按钮：**



由全局变量function\_num记录当前正在使用的功能，在鼠标点击事件***mousePressEvent（）***中使用switch(function\_num)对每次点击进行分类，根据当前功能号判断此次鼠标点击的目的是什么，并执行对应操作。而删减改点和边主要是对结点数组，边数组和距离矩阵进行对应的处理。功能号与***mousePressEvent（）***中的操作的对应关系如下：

function\_num：

1.新增结点

* + 判断点击是否有效
  + 新结点添加进结点，结点编号，结点信息数组
  + 信息栏显示

2.新增边（选择起点）

* + 确定起点，存起点数据，移交控制权至功能号3

3.新增边（选择终点）

* + 确定边有效，移交控制权回2
  + 若该边此前未添加，则加入边数组中
  + 信息栏显示

4.删除结点（选择终点）

* + 点，点编号，点信息数组中删除点后数据前移一位
  + 距离矩阵删除点后行和列前移，删除最后一行一列
  + 遍历每条边删除边（处理每条边的结点，删除点后的结点前移一位）
  + 信息显示

5.删除边（选择起点）

* + 确定起点，存起点数据，移交控制权至功能号6

6. 删除边（选择终点）

* + 遍历边数组查找是否存在匹配删除边
  + 删除边（距离矩阵对应位置0）（边数组前移）
  + 信息显示

7.编辑结点

* + 遍历结点，弹出输入框
  + 接收输入值并修改结点信息

8.编辑边（选择起点）

* + 确定起点，存起点数据，移交控制权至功能号9

9.编辑边（选择终点）

* + 遍历边数组，查找匹配边
  + 若边存在，弹出输入框
  + 接收输入值修改边的权值

## 3.文件的保存和读取

要保存下地图的所有信息，主要有三部分：结点，边和背景图。新建一个文件，利用C++的文件流操作向文件中写入如下内容：

1. 遍历结点数组，将每个结点的坐标，结点编号，结点信息输入进去，以空格隔开。
2. 将距离矩阵写入进新文件。
3. 遍历边数组，将每条边的起点，重点和权重写入新文件
4. 将地图的文件路径写入新文件。

文件读取时按顺序依次读取出文件内容并赋值给结点数组，边数组，距离句子和图片路径变量。

## 4.零碎的基础函数

1. **Recover**()按键重置：

两步操作，一个是用setText（）将所有按键的文字重置，一个是用setStyleSheet将所有按键的样式重置。

2. **Count\_distanse**(QPoint p\_1, QPoint p\_2)计算两点间距离：

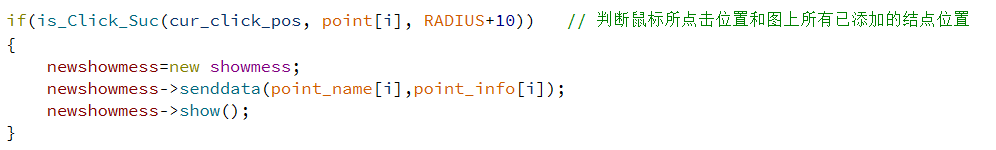
取出p\_1，p\_2的x和y坐标用距离公式求得即可。

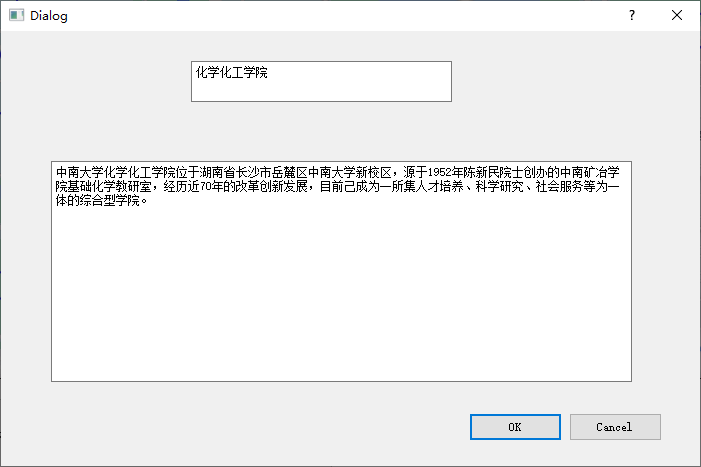
3. **is\_Click\_Suc**(QPoint x, QPoint y, int n)判断是否点击中某个点：

用**Count\_distanse**计算x,y间的距离，判断距离是不是小于n，若小于则判断点击中了。

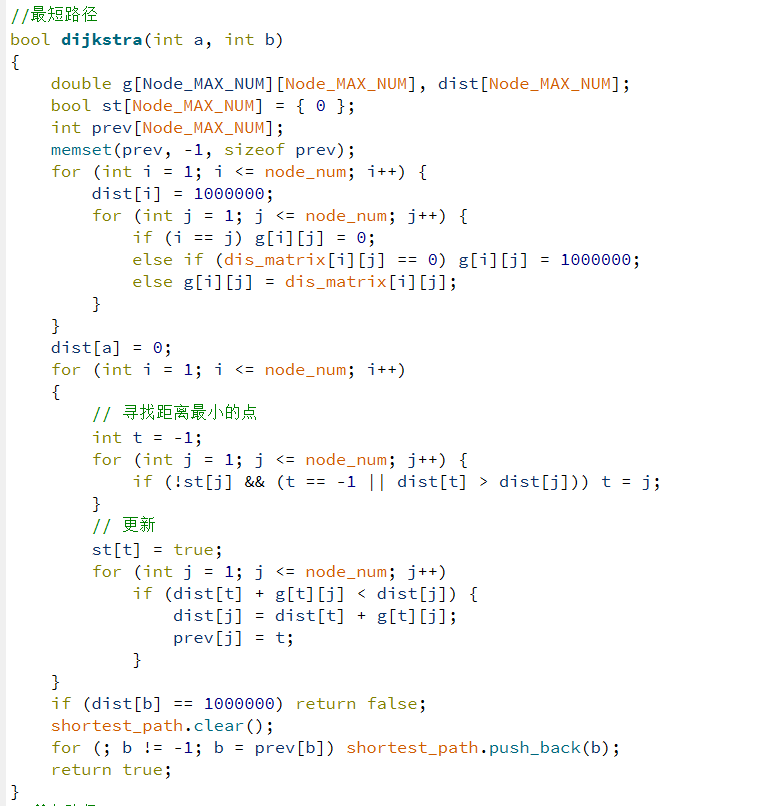
4.**mousePressEvent(QMouseEvent\* e)展示结点信息：**

获取鼠标点击位置，遍历结点匹配判断鼠标是否选中某结点，若点中了则取出该结点信息，创建新窗口并将结点信息展示出来。





## 5.dijkstra和prim最小生成树算法



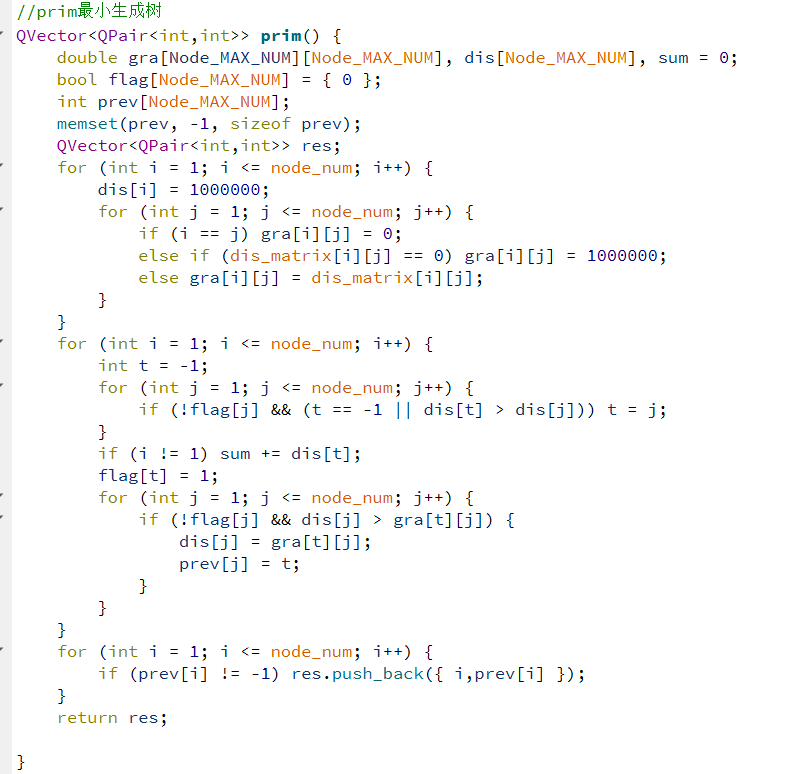
Dijkstra的基本思想是将结点分成两个集合，S集合和T集合，S集合存储已经确定最短路径的点。更新过程中不断地将T集合中距离起点最近的点添加进S集合中，并用新添加的点对T集合的点到起点的最短距离进行优化更新，即比较dist[t] + g[t][j] < dist[j]。不断重复上述过程直到所有点都添加到S集合中。

第一步先初始化算法所需的变量：二维矩阵g为距离矩阵，自身到自身为0，不可达为无穷大，故需从dis\_matrix矩阵转换一下；dist数组第i位存储当前更新状态下从a到i的最短距离，不可达则为无穷大；st作为标记数组，表示第i位的最短路径是否确定；由于要求出最短路径，故最后还需要用prev数组存储最短路径的各个结点。

第二步遍历结点数组找到未添加进S集合（即st[j]==0）的结点中距离起点最短的点（即dist[j]最小）。找到后将其添加进S集合中（st[j]置1）。

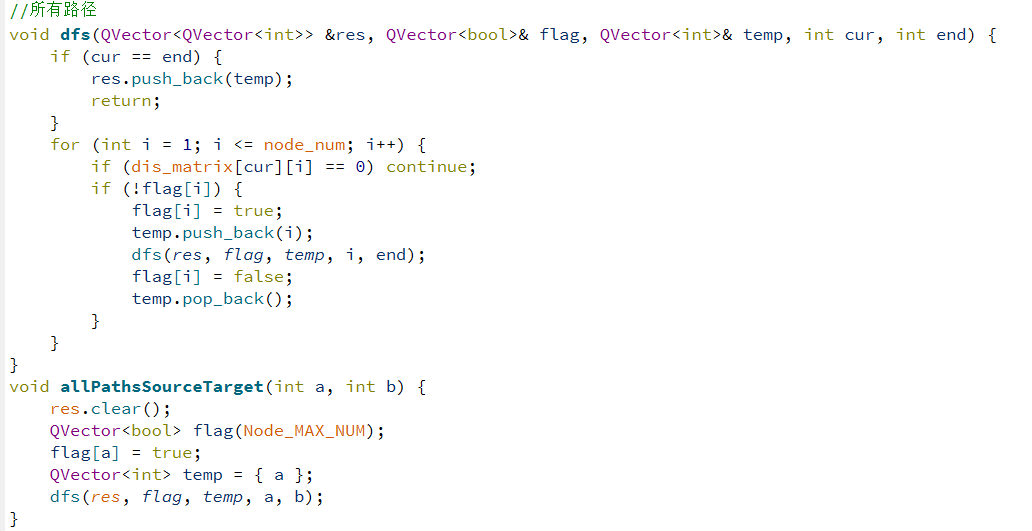
第三步利用新添加的点通过比较dist[t] + g[t][j] < dist[j]对dist数组进行更新。

第四步依次将终点的前驱结点添加进shortest\_path中，保存最短路径



Prim最小生成树算法和Dijkstra大同小异，只需在Dijkstra算法的基础上稍加修改即可。S集合表示最小生成树，T集合存未添加进最小生成树中的结点。Dist中第i位表示第i个结点到S集合的最短距离（及第i个结点距离S集合中任意点的距离最小值）。每次取出T集合中dist值最小的添加进S集合并用新添加的结点最dist进行更新（更新条件为dis[j] > gra[t][j]）。不断重复此操作直至所有结点都添加进S集合。

## 6.回溯法求所有路径



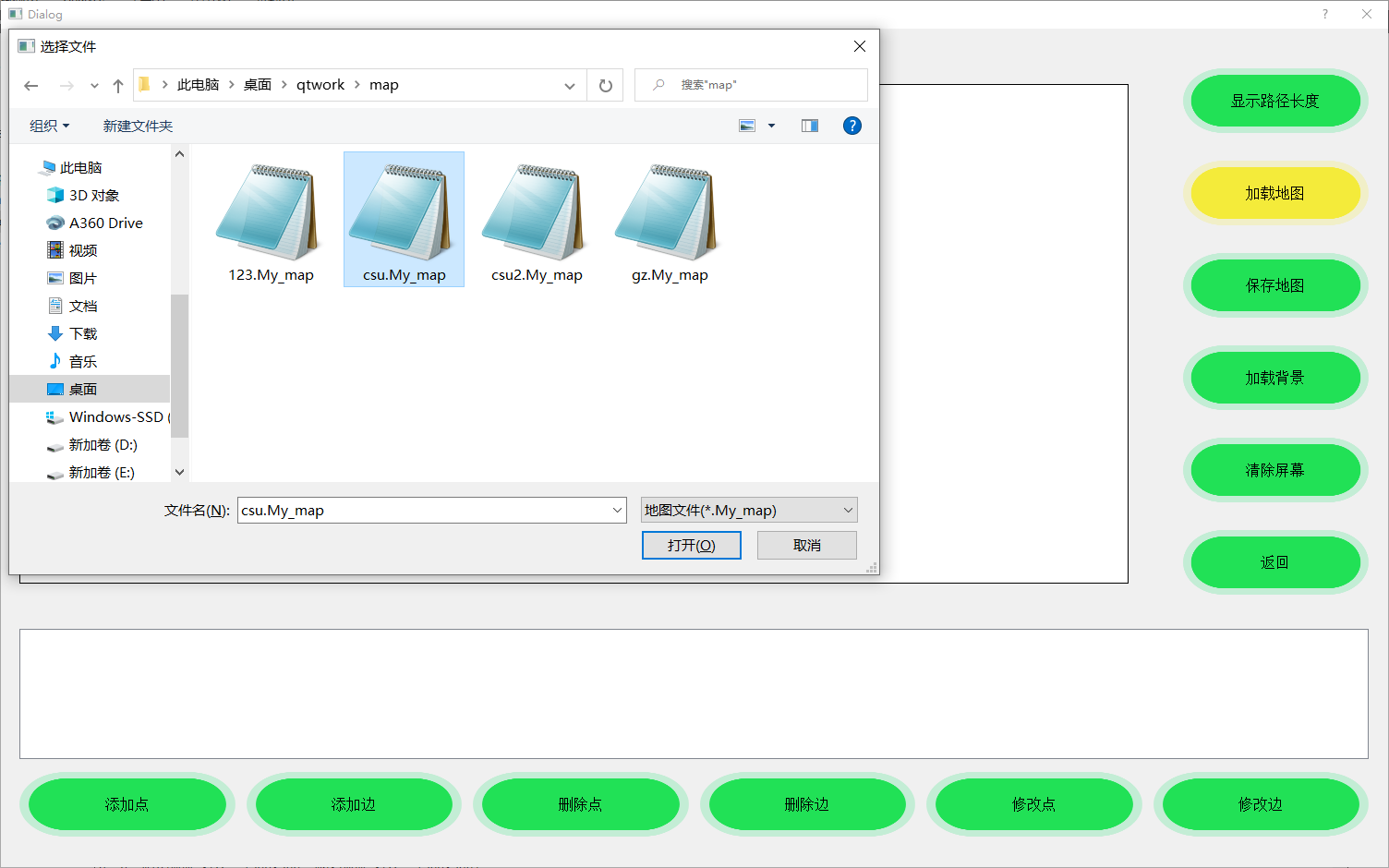
求所有路径采用回溯法，递归函数dfs中传递参数为所有路径的二维数组，结点访问标记数组，当前路径数组，当前位置，终点位置。递归终止条件为当前位置等于终点位置，即当前路径为一条所求路径，将此时的当前路径数组添加进所有路径中。递归函数中遍历所有结点找到未访问的结点后结点访问标记数组对应位置1，添加进当前路径数组，递归，然后恢复现场（即结点访问标记数组对应位置0，删除当前路径数组最后一位）。

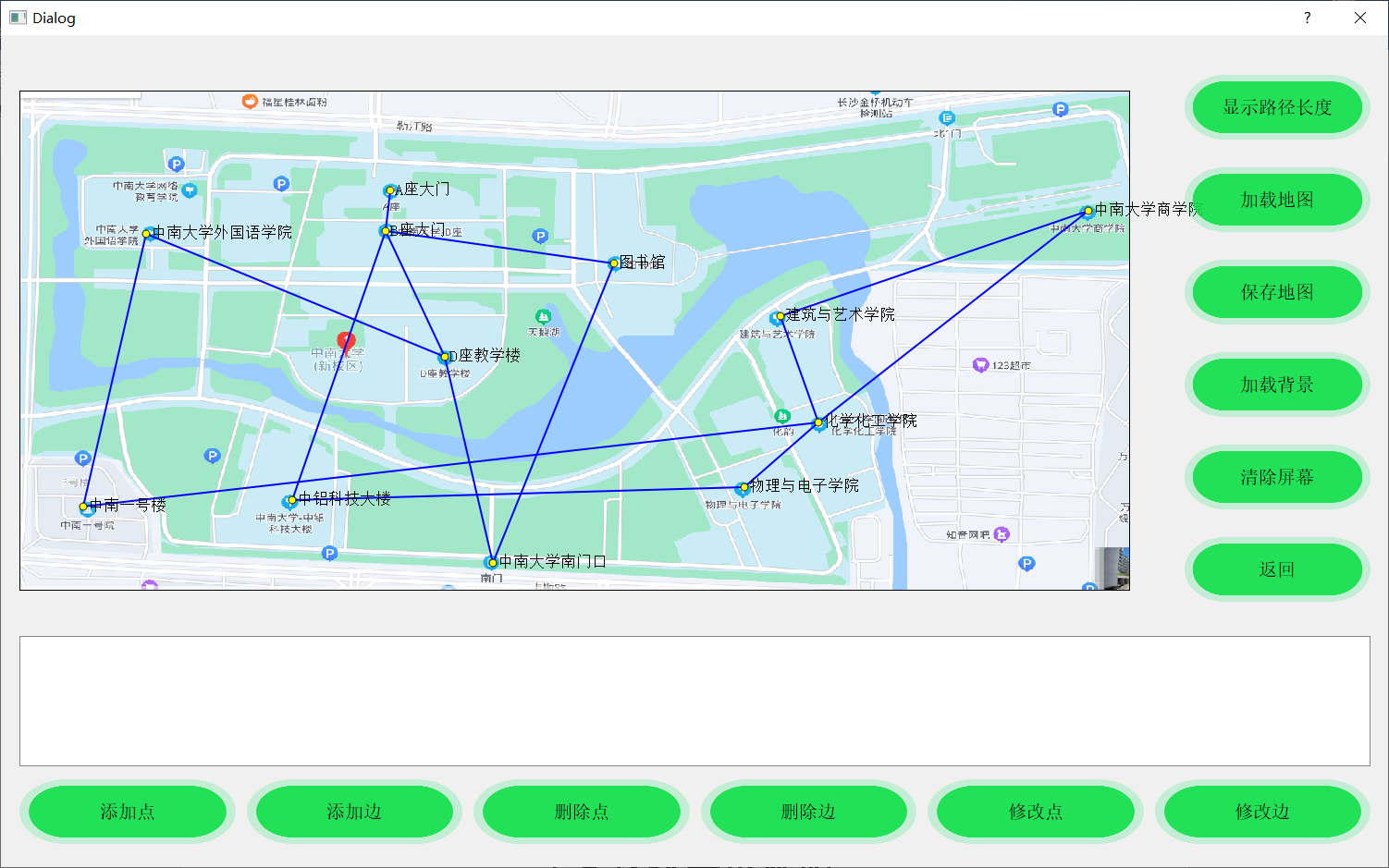
# 4.运行结果

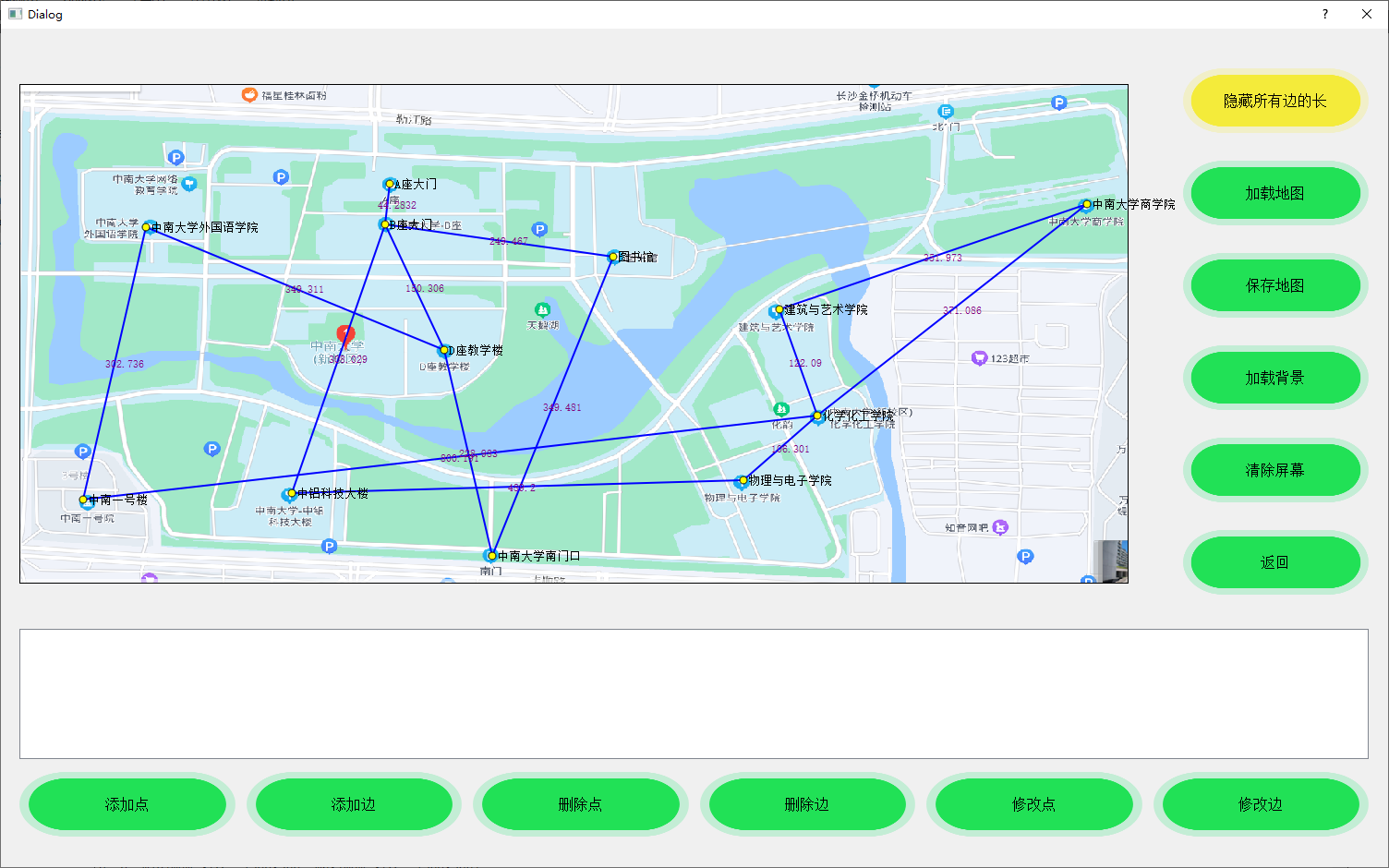
## 主界面：

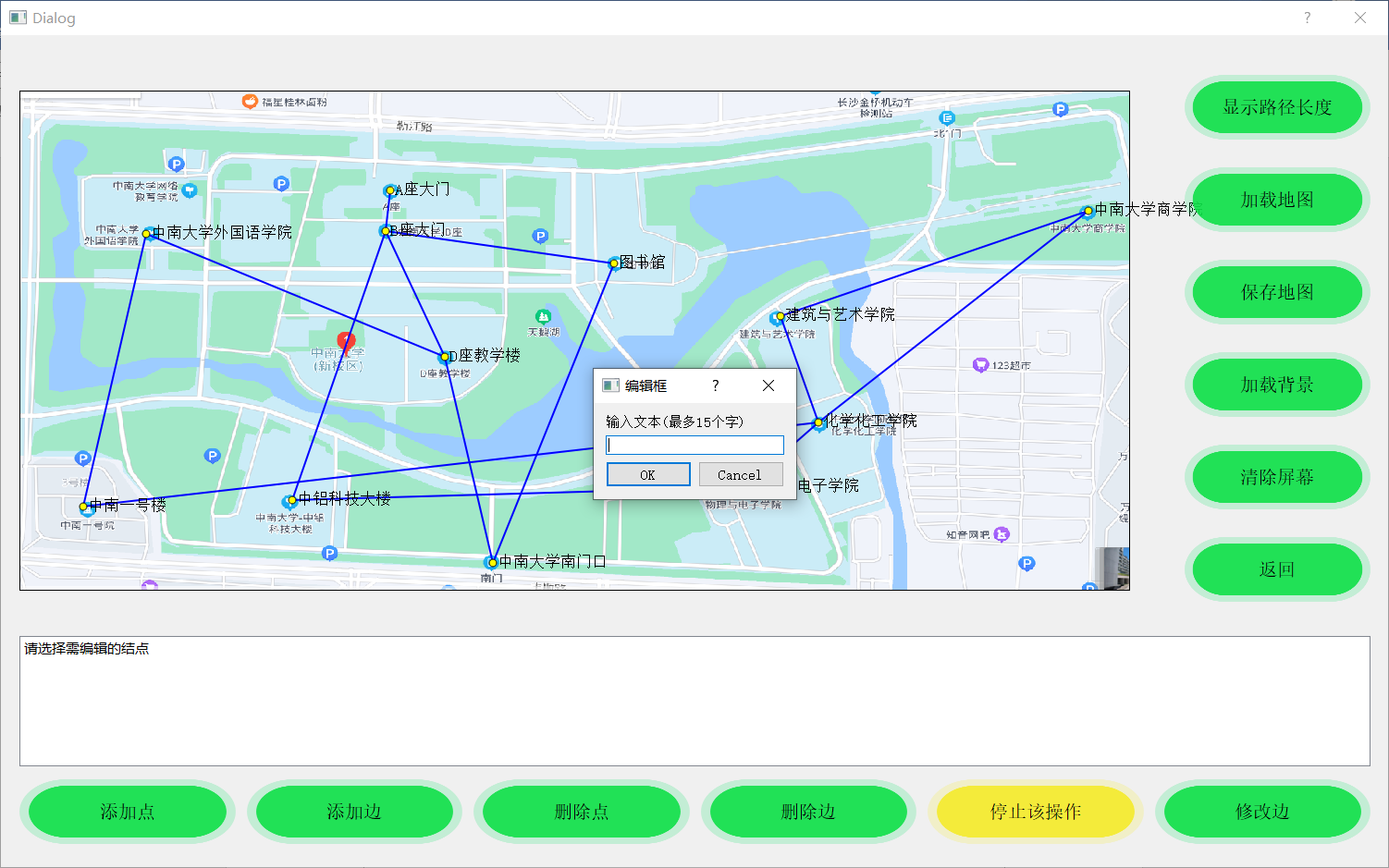


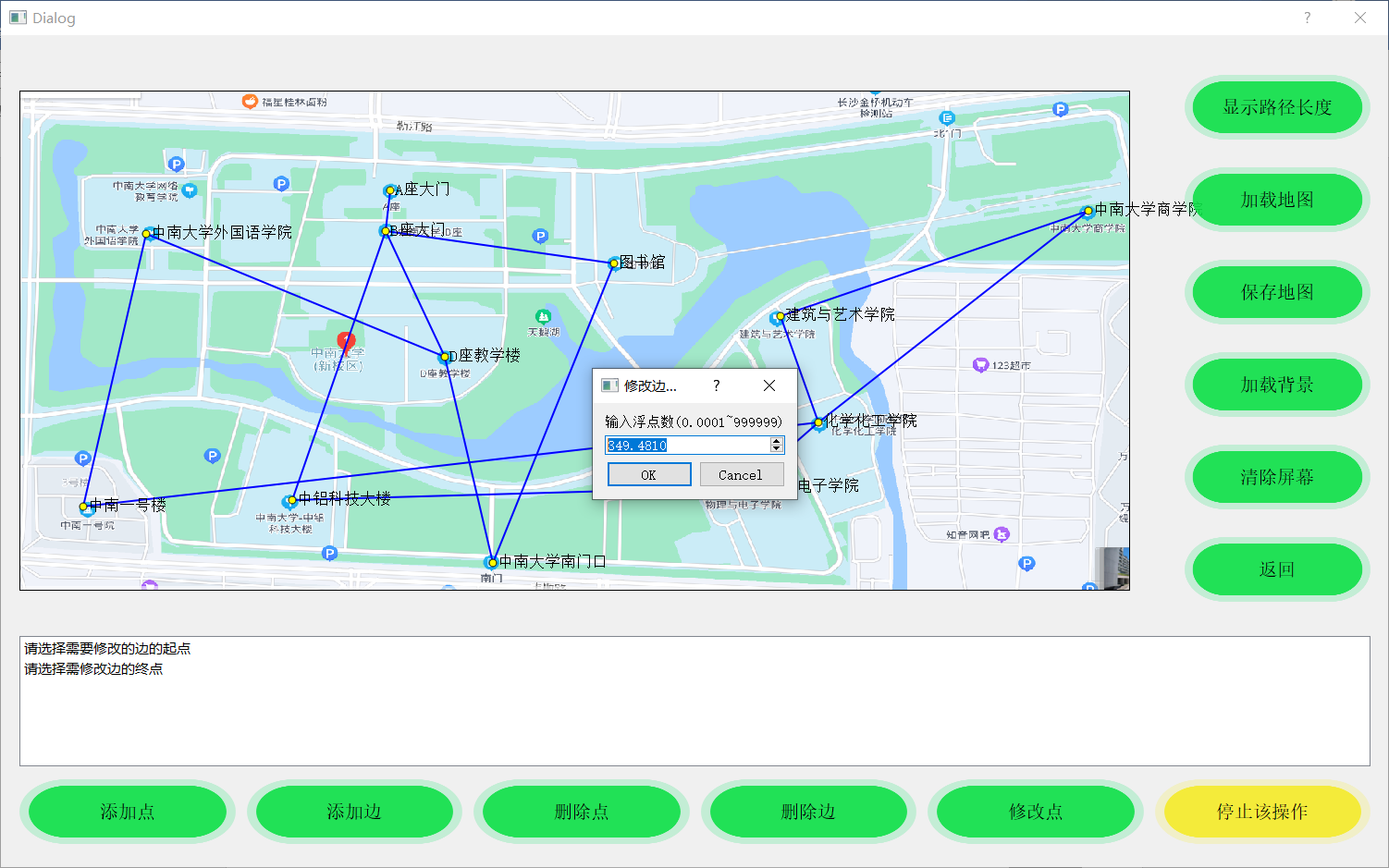
## 地图制作界面

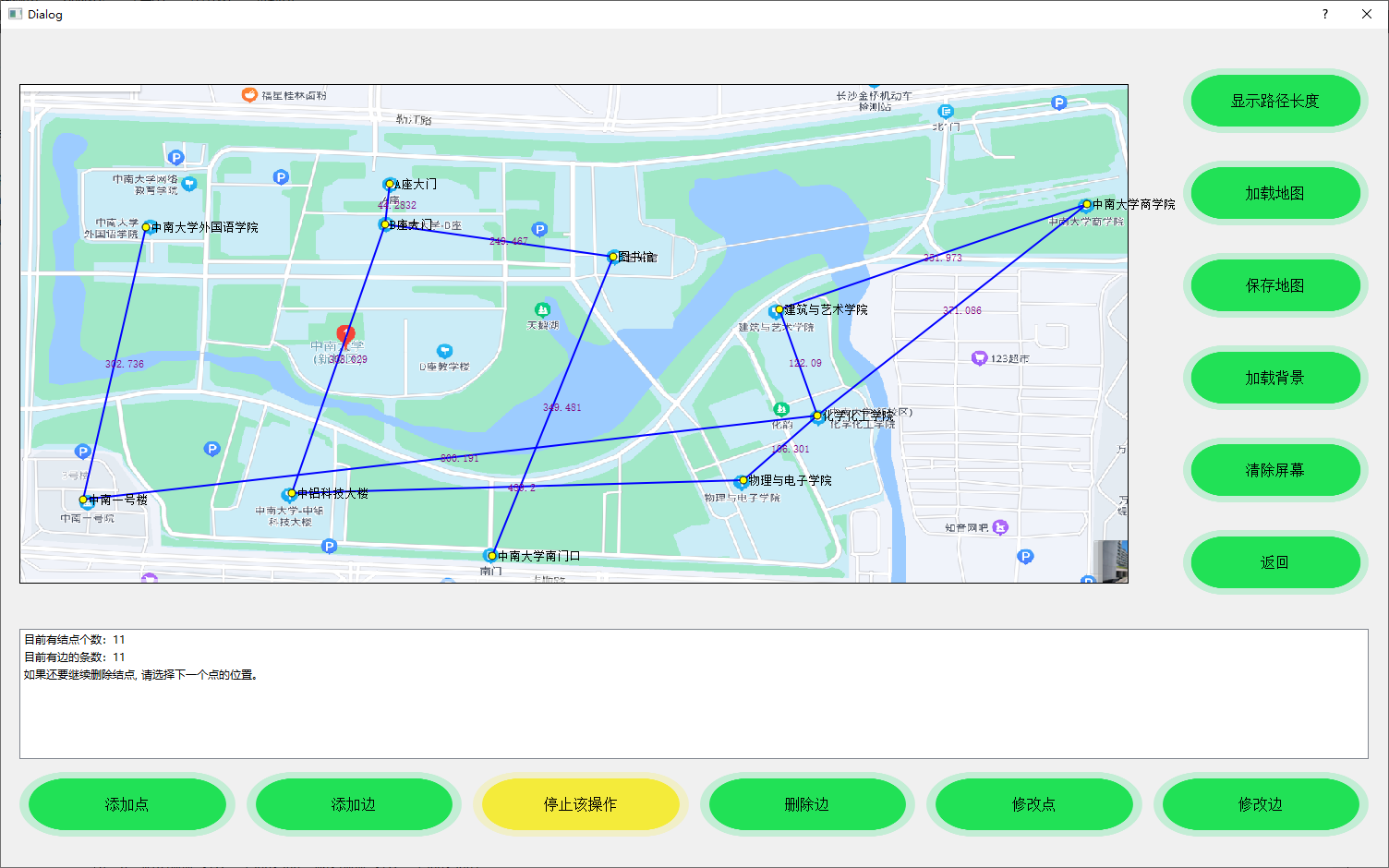


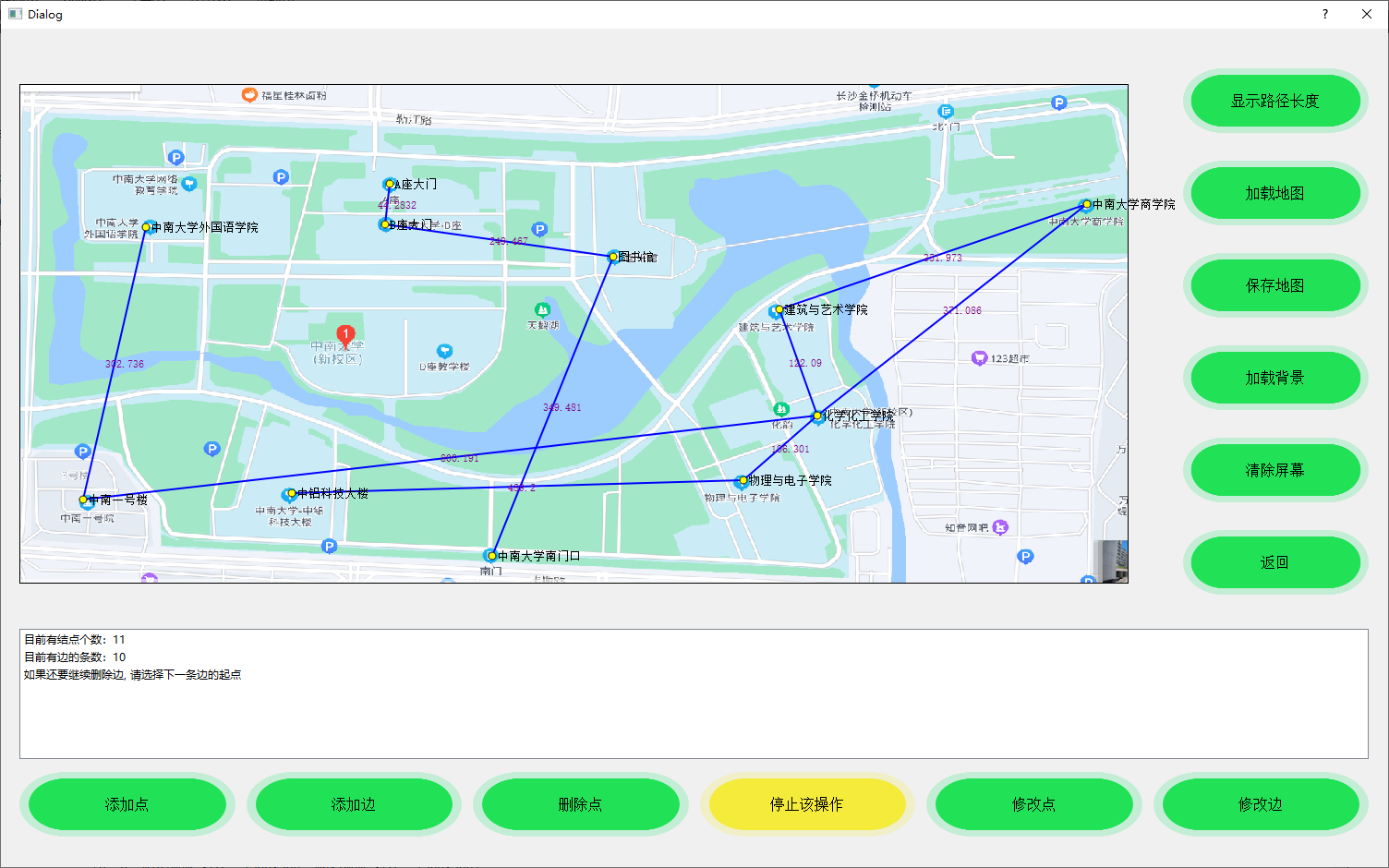


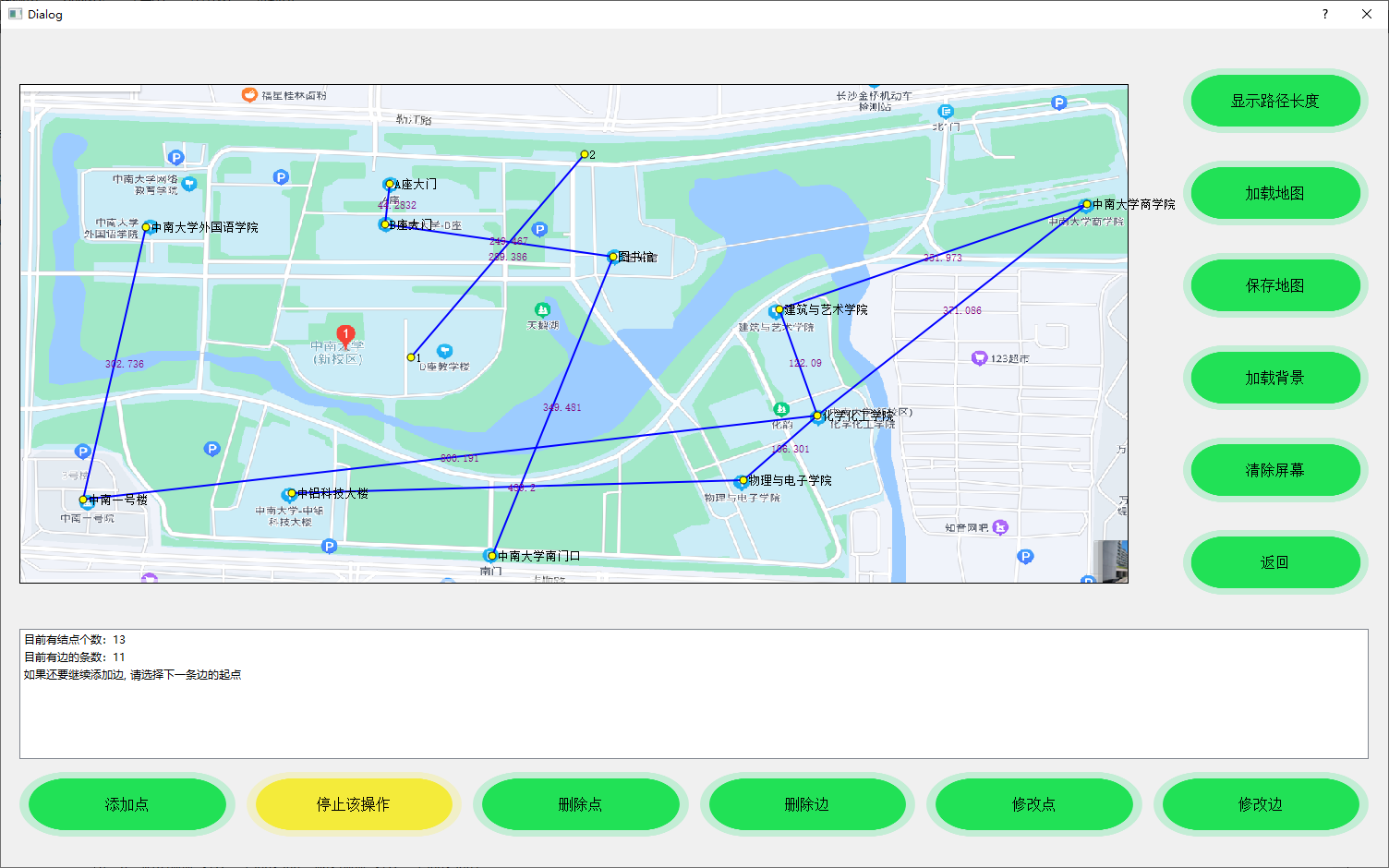




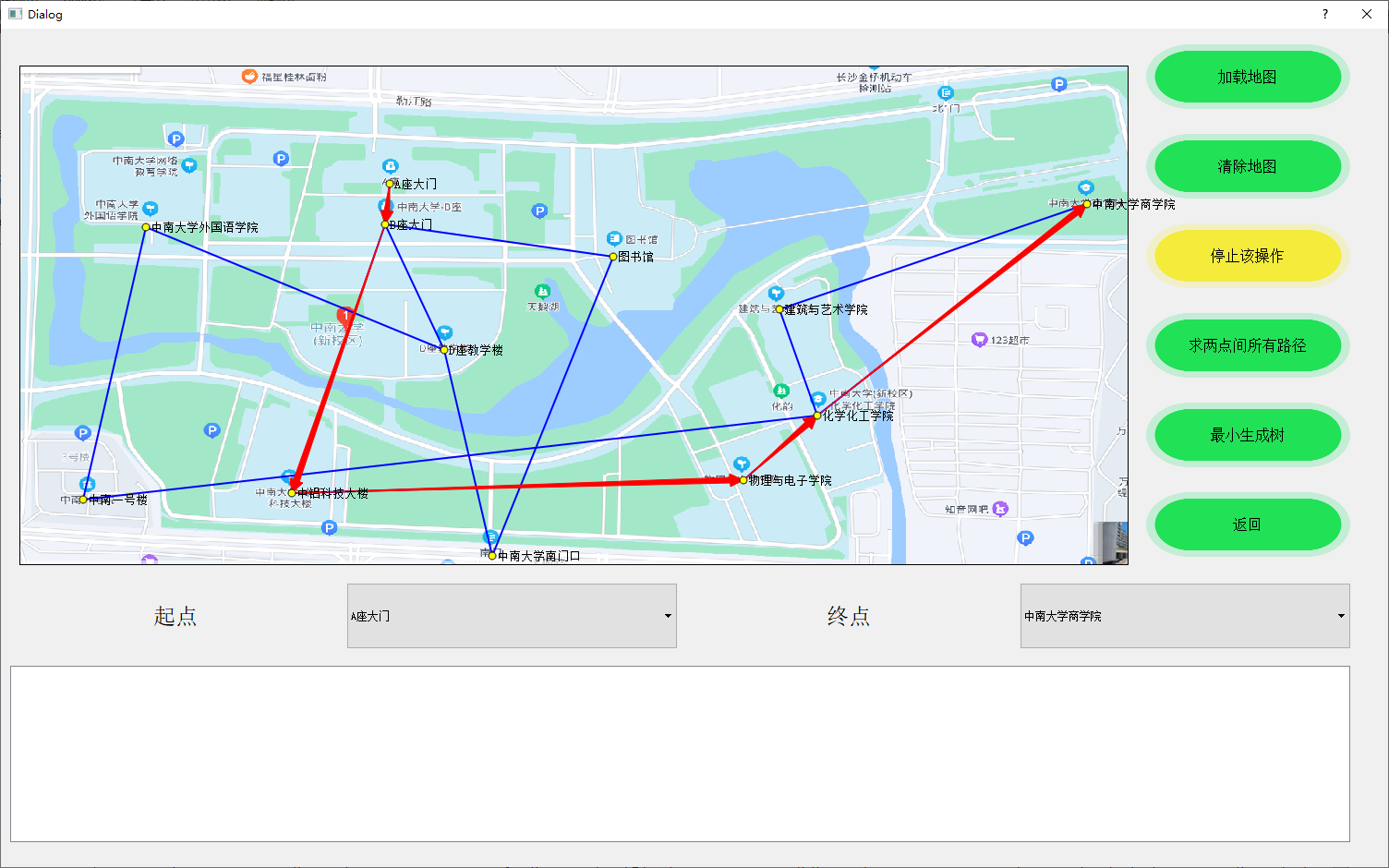


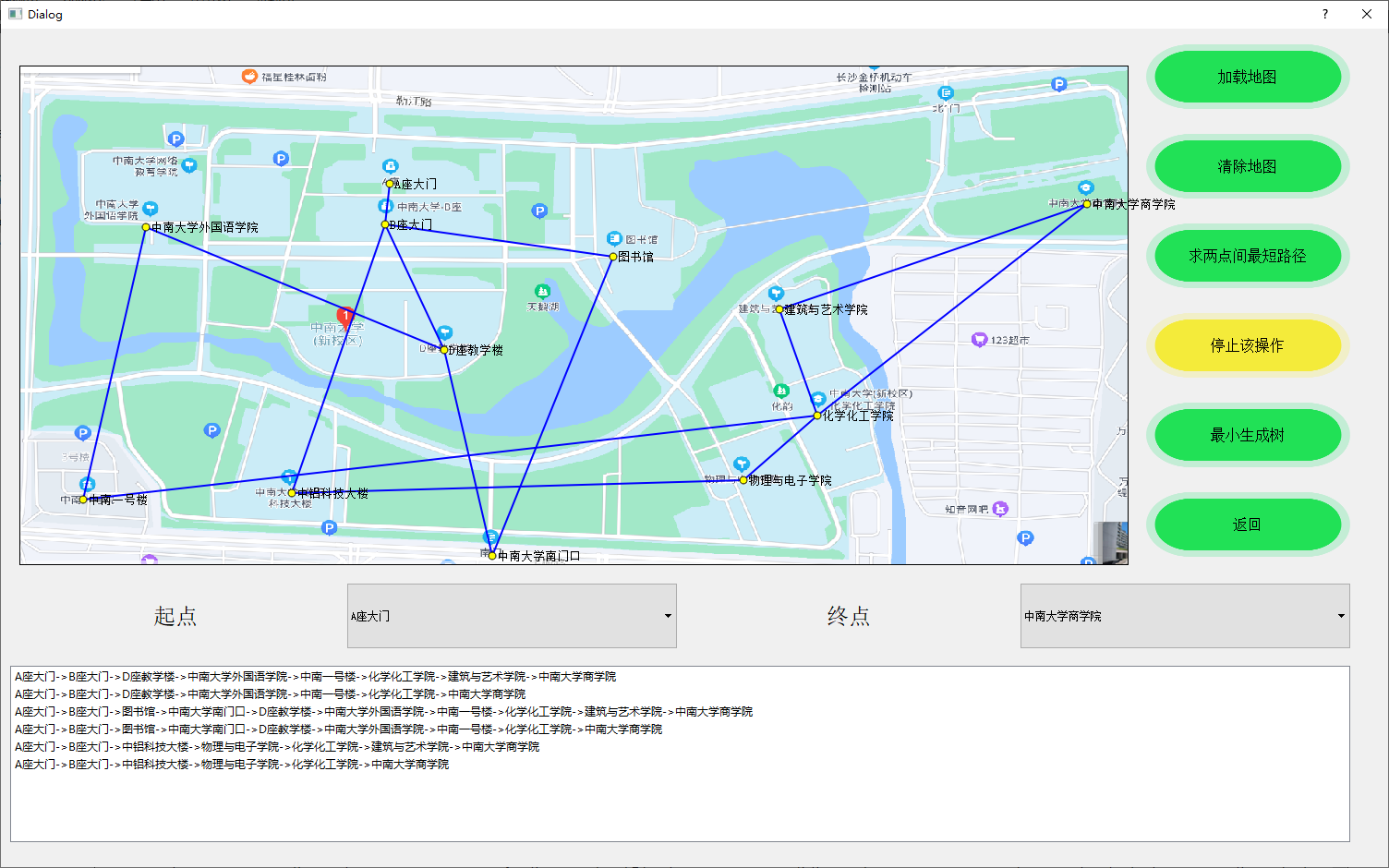


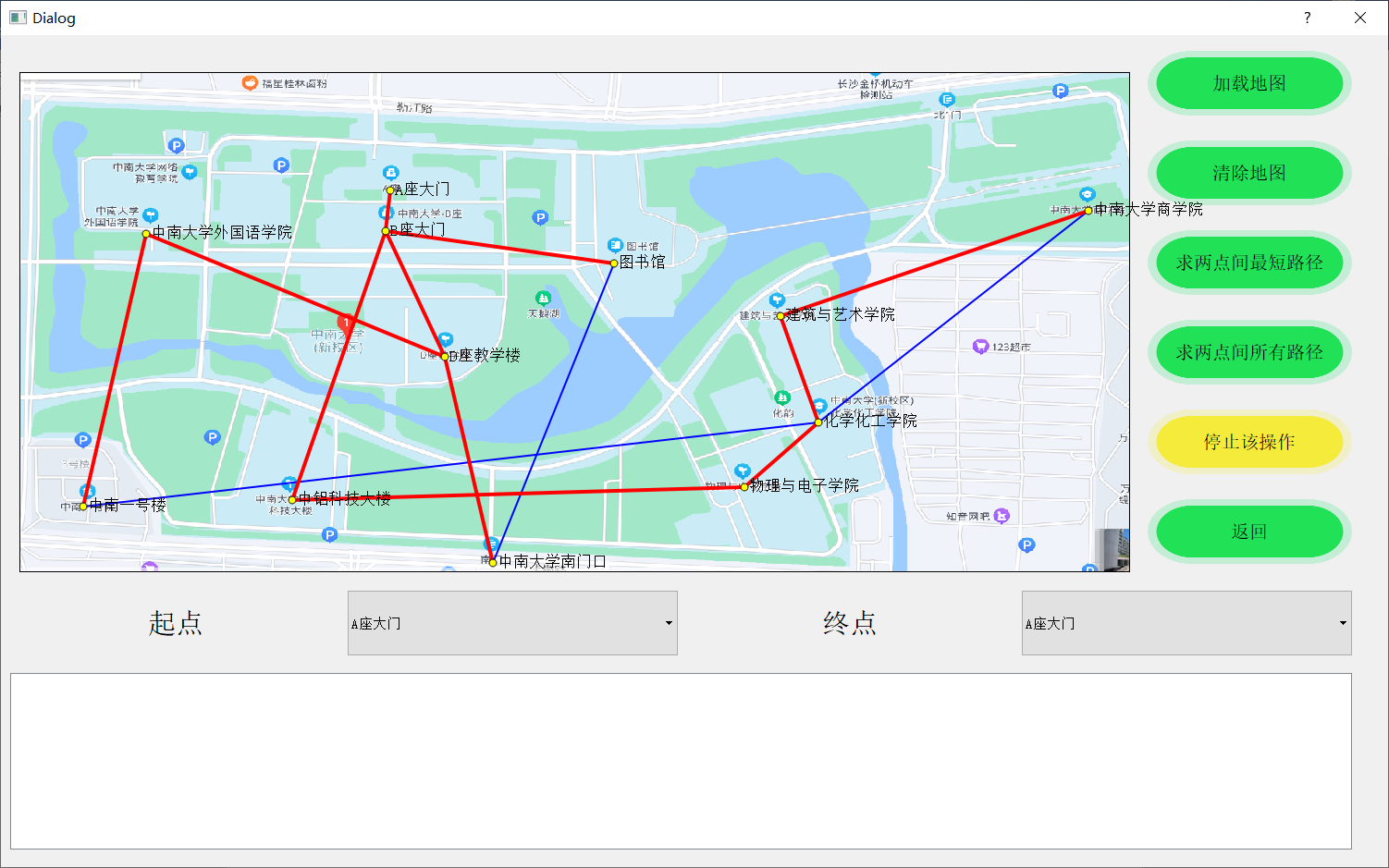


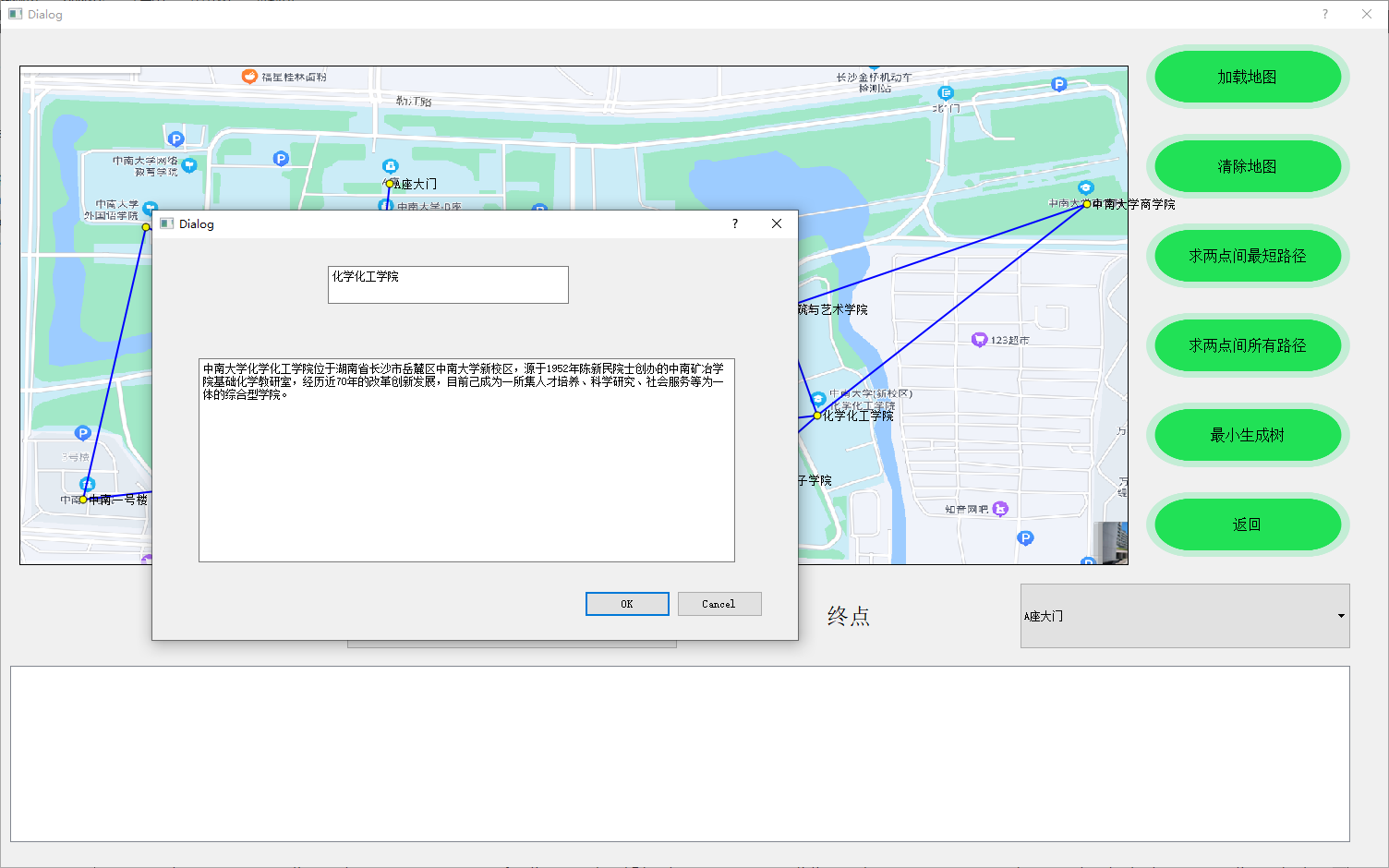


## 地图查询界面：









# 5.课程设计总结

1.项目编程：

项目编程不像过去只敲一个文件这么简单，涉及到了多个文件之间的关联。过去对static的理解并不深刻，但这次因为两个文件中用到了相同的变量名编译时疯狂报错，变量重定义。经过重重检查和网上搜索后发现是两个文件中的同名变量出了问题，加了static后就顺利编译了。实践中切身地感受到了项目编程中文件结构的重要性，头文件申明类，cpp文件实现类，ui文件实现界面，开一个新文件夹专门用来存图片等，这样有条理地组织各种文件能使敲代码时思路更清晰，结构更严谨，不容易出错。

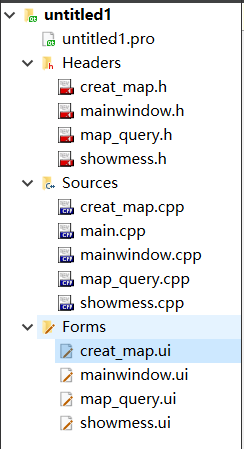
2.算法思想：

亲手实践了dijkstra和prim以及搜索所有路径的算法，在网上浏览各种算法代码编写方式的过程中加深了对这几种算法的理解，同时在编写各种函数，各种函数与程序中变量的交互中感受到了函数封装性的重要，一个函数提供一个接口，返回一个结果，尽量与程序中的变量抛开关系，将功能模块化，写函数时会更舒服，检查错误时也更方便。

3.善于利用搜索引擎，勤做笔记：

在互联网这种更新迭代速度飞快的行业，作为程序员，完全掌握一种语言，一项软件，记住里面的各种函数是很难的也不必要的，语言和软件会不断更新，函数多得背不完，同时制作一个项目的过程中就会接触到各种新的语言，软件，是来不及完全学习的。所以我们要锻炼的是一种学习能力，搜索能力，和记录能力。深刻理解好一门语言，一类软件之类的思想，以后接触其他的便会触类旁通。此外更要善于上网搜索，刚开始接触QT的时候我也是一脸茫然不知所措，但随着在网上看了一些文章，同时记录下常见的各种QT内置的特殊函数，诸如各种绘图函数，慢慢地也就开始掌握了。写代码时遇到各种错误不一定要盯着自己的代码仔细检查，可以先复制报错信息去网上查，很多我遇到的问题，其他人也会遇到并记录下来，比起一个人瞪着代码纠错，效率要高不少。此外，面对一项新的软件，会出现很多陌生的函数，短时间内很难记住，这个时候应该及时地将函数的入参，返回值，功能记录在专门的笔记文档中，而不是每次遇到都去查一遍，会浪费很多时间。

# 附录



## Untitled1.pro

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use

# any Qt feature that has been marked deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

creat\_map.cpp \

main.cpp \

mainwindow.cpp \

map\_query.cpp \

showmess.cpp

HEADERS += \

creat\_map.h \

mainwindow.h \

map\_query.h \

showmess.h

FORMS += \

creat\_map.ui \

mainwindow.ui \

map\_query.ui \

showmess.ui

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

## Creat\_map.h

#ifndef CREAT\_MAP\_H

#define CREAT\_MAP\_H

#include <QDialog>

namespace **Ui** {

class **creat\_map**;

}

class **creat\_map** : public QDialog

{

Q\_OBJECT

public:

explicit **creat\_map**(QWidget \*parent = nullptr);

~***creat\_map***();

private slots:

void ***paintEvent***(QPaintEvent\*); /\* 窗口绘制 \*/

void ***mousePressEvent***(QMouseEvent\* event); /\* 鼠标响应 \*/

void **Recover**();

double **Count\_distanse**(QPoint p\_1, QPoint p\_2);

bool **is\_Click\_Suc**(QPoint x, QPoint y, int n);

void **on\_back\_clicked**();

void **on\_add\_node\_clicked**();

void **on\_add\_edge\_clicked**();

void **on\_del\_node\_clicked**();

void **on\_del\_edge\_clicked**();

void **on\_mod\_node\_clicked**();

void **on\_mod\_edge\_clicked**();

void **on\_showlen\_clicked**();

void **on\_load\_map\_clicked**();

void **on\_save\_map\_clicked**();

void **on\_load\_bg\_clicked**();

void **on\_clear\_bg\_clicked**();

private:

Ui::creat\_map \*ui;

};

#endif // CREAT\_MAP\_H

## Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_2\_clicked**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

## Map\_query

#ifndef MAP\_QUERY\_H

#define MAP\_QUERY\_H

#include <QDialog>

#include"showmess.h"

namespace **Ui** {

class **map\_query**;

}

class **map\_query** : public QDialog

{

Q\_OBJECT

public:

explicit **map\_query**(QWidget \*parent = nullptr);

~***map\_query***();

private slots:

void ***paintEvent***(QPaintEvent\*); /\* 窗口绘制 \*/

void ***mousePressEvent***(QMouseEvent\* event); /\* 鼠标响应 \*/

double **Count\_distanse**(QPoint p\_1, QPoint p\_2);

bool **is\_Click\_Suc**(QPoint x, QPoint y, int n);

void **Recover**();

void **on\_back\_clicked**();

void **on\_load\_map\_clicked**();

void **on\_shortest\_rode\_clicked**();

void **on\_all\_rode\_clicked**();

void **on\_mintree\_clicked**();

void **on\_clear\_clicked**();

private:

Ui::map\_query \*ui;

showmess\* newshowmess;

};

#endif // MAP\_QUERY\_H

## Showmess.h

#ifndef SHOWMESS\_H

#define SHOWMESS\_H

#include <QDialog>

namespace Ui {

class showmess;

}

class showmess : public QDialog

{

Q\_OBJECT

public:

explicit showmess(QWidget \*parent = nullptr);

~showmess();

void senddata(QString s1,QString s2);

private:

Ui::showmess \*ui;

};

#endif // SHOWMESS\_H

## Creat\_map.cpp

#include "creat\_map.h"

#include "ui\_creat\_map.h"

#include "mainwindow.h"

#include <QtWidgets>

#include <vector>

#include <fstream>

#include <algorithm>

typedef struct **line\_struct** // 边的结构体

{

int ind; // 每条边的编号

int node\_1, node\_2; // 边的起点和终点

double dis; // 边的长度

}Line;

static const int show\_window\_x = 20; // 窗口的左上角x

static const int show\_window\_y = 60; // 窗口的左上角y

static const int show\_window\_width = 1200; // 窗口的宽

static const int show\_window\_height = 540; // 窗口的高

static const int Node\_MAX\_NUM = 100; // 设置最大结点数

static const int Side\_MAX\_NUM = 300; // 设置最大边数

static const int RADIUS = 5; // 点击半径

static int function\_num = 0; // 功能号(针对不同的操作会产生不同的效果)

static int node\_num; // 结点个数

static int side\_num; // 边的条数

static int info\_ind; // 结点标签的数组下标

static QPoint point[Node\_MAX\_NUM]; // 结点数组

static Line line[Side\_MAX\_NUM]; // 边数组

static double dis\_matrix[Node\_MAX\_NUM][Node\_MAX\_NUM]; // 距离矩阵

static QString point\_name[Node\_MAX\_NUM]; // 结点的文字标签数组

static QString point\_info[Node\_MAX\_NUM]; // 结点的信息标签数组

static QString pic\_path; // 图片路径

static bool show\_len\_flag = false; // 是否显示当前路径长度

//构造函数

creat\_map::**creat\_map**(QWidget \*parent) :QDialog(parent),ui(new Ui::creat\_map)

{

ui->setupUi(this);

Recover();

}

//析构函数

creat\_map::~***creat\_map***()

{

delete ui;

}

// 按钮信息重置

void creat\_map::**Recover**()

{

// ui->Message\_1->setTextElideMode(Qt::ElideNone); // 自动换行

ui->Message\_1->clear();

ui->add\_node->setText("添加点");

ui->add\_edge->setText("添加边");

ui->del\_node->setText("删除点");

ui->del\_edge->setText("删除边");

ui->mod\_node->setText("修改点");

ui->mod\_edge->setText("修改边");

ui->back->setText("返回");

ui->clear\_bg->setText("清除屏幕");

ui->load\_bg->setText("加载背景");

ui->load\_map->setText("加载地图");

ui->save\_map->setText("保存地图");

ui->showlen->setText("显示路径长度");

ui->add\_node->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->add\_edge->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->del\_node->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->del\_edge->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->mod\_node->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->mod\_edge->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->back->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->clear\_bg->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->load\_bg->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->load\_map->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->save\_map->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->showlen->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

// 计算两点间距离的函数

double creat\_map::**Count\_distanse**(QPoint p\_1, QPoint p\_2)

{

double delta\_x, delta\_y;

delta\_x = p\_1.x() - p\_2.x();

delta\_y = p\_1.y() - p\_2.y();

return sqrt( delta\_x\*delta\_x + delta\_y\*delta\_y );

}

// 判断目标是否选中的函数

bool creat\_map::**is\_Click\_Suc**(QPoint x, QPoint y, int n) // 判断目标是否选中的函数

{

if (Count\_distanse(x, y) <= n)

return true;

else

return false;

}

//绘制函数

void creat\_map::***paintEvent***(QPaintEvent\*){

QPainter painter(this); // 画笔，this为当前窗口

QFont font1("ZYSong 18030", 8); // 新建 1 号字体

QFont font2("ZYSong 18030", 10); // 新建 2 号字体

this->setMaximumSize(1500,900); // 固定大小

this->setMinimumSize(1500,900);

if( pic\_path == "" ) // 如果无背景图片

painter.setBrush(Qt::white); // 白底填充

else

{

painter.drawPixmap(show\_window\_x, show\_window\_y, show\_window\_width, show\_window\_height, QPixmap(pic\_path)); // 载入背景图片并显示出来

}

painter.drawRect(show\_window\_x, show\_window\_y, show\_window\_width, show\_window\_height); // 绘制一个矩形框

painter.setFont(font2);

painter.setRenderHint(QPainter::Antialiasing, true); // 设置:绘图光滑

//画线

for( int i = 1; i <= side\_num; i++ ) // 先画线，更好看！不然线有覆盖“景点”的情况出现

{

painter.setPen(QPen(Qt::blue,2)); // 设置画笔颜色为蓝色

painter.drawLine(point[line[i].node\_1], point[line[i].node\_2]); //两点连线

}

//标边长

painter.setFont(font1);

if(show\_len\_flag) //如果显示边的标签

{

painter.setPen(Qt::darkMagenta); // 洋红色

for( int i = 1; i < node\_num; i++ )

{

for( int j = i + 1; j <= node\_num; j++ )

{

if( dis\_matrix[i][j] ) //若路径存在

{

int zuo\_x, zuo\_y;

QString str = QString::number(dis\_matrix[i][j]); // 标签

zuo\_x = (point[i].x() + point[j].x()) / 2 - 10;

zuo\_y = (point[i].y() + point[j].y()) / 2 + 5;

painter.drawText( QPoint(zuo\_x, zuo\_y), str ); // 写在该“边”的中间偏左下的地方

}

}

}

}

//标点

painter.setFont(font2);

for(int i = 1; i <= node\_num; i++)

{

painter.setPen(Qt::black); // 黑色

painter.setBrush(Qt::yellow); // 内部填充(黄色)

painter.drawEllipse(point[i], 4, 4); // 把点画出来

painter.drawText(QPoint(point[i].x() + 5, point[i].y() + 5), point\_name[i]); // 画出点的标签，point\_name[i]为标签

}

}

Line temp\_Line; // 临时边

QPoint temp\_Point\_1, temp\_Point\_2; // 两个临时的点

QString tmp1,tmp2;

//鼠标响应

void creat\_map::***mousePressEvent***(QMouseEvent\* e){

if (e->button() == Qt::LeftButton) // 按左键

{

QPoint cur\_click\_pos = e->pos(); // e->pos(): 获取当前点击位置

switch(function\_num)

{

case 1: // 添加点

if(node\_num < Node\_MAX\_NUM && cur\_click\_pos.x() >= show\_window\_x &&

cur\_click\_pos.x() <= show\_window\_x+show\_window\_width && cur\_click\_pos.y() >= show\_window\_y &&

cur\_click\_pos.y() <= show\_window\_y+show\_window\_height) // 判断所加的点是否在窗口范围内

{

int save\_node\_num = node\_num;

node\_num++;

for(int i = 1; i < node\_num; i++)

{

if(is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS+10)) // 判断鼠标所点击位置和图上所有已添加的结点位置,是否靠的太近

{

node\_num--;

QMessageBox::warning(this, "警告", "两个点靠太近！");

}

}

if(save\_node\_num == node\_num)

break;

point[node\_num] = e->pos(); // 当前位置赋给最新的结点

point\_name[node\_num] = QString::number(++info\_ind); // 创建默认标签

point\_info[node\_num]="无信息";

update();

}

else if(node\_num >= Node\_MAX\_NUM) QMessageBox::warning(this, "警告", "目前结点数已达上限，无法再继续添加！");

else QMessageBox::warning(this, "警告", "新加结点已超出边界！");

ui->Message\_1->clear();

ui->Message\_1->addItem("目前有结点个数：" + QString::number(node\_num));

ui->Message\_1->addItem("目前有边的条数：" + QString::number(side\_num));

ui->Message\_1->addItem("如果还要继续添加, 请选择下一个点的位置。");

function\_num = 1; // 功能号 1 保持不变(便于重复添加点)

break;

case 2: // 新增边(选择边的起点)

if(side\_num >= Side\_MAX\_NUM) QMessageBox::warning(this, "警告", "路径数已达上限！");

else

{

for( int i = 1; i <= node\_num; i++ )

{

if( is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS) ) // 判断是否选中

{

function\_num = 3; // 找到了新增边的起点后, 还需找到其终点. 故把控制权交给功能号3

temp\_Point\_1 = point[i];

line[side\_num + 1].node\_1 = i;

ui->Message\_1->clear();

ui->Message\_1->addItem("请选择边的终点位置");

break;

}

}

}

update();

break;

case 3: // 新增边(选择边的终点)

for( int i = 1; i <= node\_num; i++ )

{

if(point[i] != temp\_Point\_1 && is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS)) // 若选中了与第一个点不同的点

{

function\_num = 2; // 重新把控制权交给功能号2(便于重复添加“边”)

int save\_side\_num = side\_num++; // 线数量 + 1

temp\_Point\_2 = point[i];

line[side\_num].node\_2 = i;

if(line[side\_num].node\_1 > line[side\_num].node\_2) // 确保边的起点下标比终点的小, 不然做交换

{

int temp = line[side\_num].node\_1;

line[side\_num].node\_1 = line[side\_num].node\_2;

line[side\_num].node\_2 = temp;

}

for( int j = 1; j < side\_num; j++ ) // 判断是否路线已经存在

{

if(line[side\_num].node\_1 == line[j].node\_1 && line[side\_num].node\_2 == line[j].node\_2)

{

line[side\_num] = line[0];

side\_num--;

QMessageBox::warning(this, "警告", "该路径已添加！");

break;

}

}

if(save\_side\_num != side\_num) // 如果路该线之前在图中不存在, 则该表达式成立

{

int ind\_1 = line[side\_num].node\_1;

int ind\_2 = line[side\_num].node\_2;

dis\_matrix[ind\_1][ind\_2] = dis\_matrix[ind\_2][ind\_1] = Count\_distanse(point[ind\_1], point[ind\_2]); // 距离矩阵赋值

line[side\_num].ind = side\_num; // 边的“编号”

line[side\_num].dis = Count\_distanse(point[ind\_1], point[ind\_2]); // 边的长度

}

ui->Message\_1->clear();

ui->Message\_1->addItem("目前有结点个数：" + QString::number(node\_num));

ui->Message\_1->addItem("目前有边的条数：" + QString::number(side\_num));

ui->Message\_1->addItem("如果还要继续添加边, 请选择下一条边的起点");

break;

}

}

update();

break;

case 4: // 删除结点

if(node\_num == 0)

{

QMessageBox::warning(this, "警告", "没有结点可删除！");

}

else

{

for( int i = 1; i <= node\_num; i++ )

{

if(is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS))

{

for( int j = i; j < node\_num; j++ )

{

point[j] = point[j + 1]; // 将删除的结点后的点前移

point\_name[j] = point\_name[j + 1]; // 结点的标签做同样处理

point\_info[j] = point\_info[j + 1];

}

// 删除该结点所在的行

for( int j = i; j < node\_num; j++ )

for ( int k = 1; k <= node\_num; k++ )

dis\_matrix[k][j] = dis\_matrix[k][j + 1];

// 删除该结点所在的列

for( int j = i; j < node\_num; j++ )

for( int k = 1; k <= node\_num; k++ )

dis\_matrix[j][k] = dis\_matrix[j + 1][k];

//删除最后一行一列

for (int j = 1; j <= node\_num; j++)

dis\_matrix[j][node\_num] = dis\_matrix[node\_num][j] = 0;

point[node\_num] = point[0]; // 最后一个点消失

point\_name[node\_num] = point\_name[0]; // 最后一个点标签消失(注意:其他点的标签保持不变)

point\_info[node\_num] = point\_info[0];

node\_num--;

for( int j = 1; j <= side\_num; j++ ) // 对每一边都进行相应的判断

{

if(i == line[j].node\_1 || i == line[j].node\_2) // 只要边的起点和终点有一个是“即将被删除的结点”,则进行后续处理

{

for( int k = j; k < side\_num; k++ ) // 将“边数组”也一并前移【除ind】

{

line[k].dis = line[k + 1].dis;

line[k].node\_1 = line[k + 1].node\_1;

line[k].node\_2 = line[k + 1].node\_2;

}

line[side\_num] = line[0]; // 将最后一条边去除

side\_num--;

j--; // 连续判断所需(因为边数组已前移)

}

else // 在“边数组”里面, 也要更新每一条边(大于删除结点编)的起点和终点

{

if(line[j].node\_1 > i)

line[j].node\_1--;

if(line[j].node\_2 > i)

line[j].node\_2--;

}

}

update();

break;

}

}

}

ui->Message\_1->clear();

ui->Message\_1->addItem("目前有结点个数：" + QString::number(node\_num));

ui->Message\_1->addItem("目前有边的条数：" + QString::number(side\_num));

ui->Message\_1->addItem("如果还要继续删除结点, 请选择下一个点的位置。");

function\_num = 4; // 功能号 4 保持不变(便于重复进行删除操作)

break;

case 5: // 删除边(选择边的起点)

if(side\_num == 0) QMessageBox::warning(this, "警告", "图中已无边可删除！");

else

{

for( int i = 1; i <= node\_num; i++ )

{

if(is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS))

{

function\_num = 6; // 找到了删除边的起点后, 还需找到其终点. 故把控制权交给功能号6

temp\_Point\_1 = point[i];

temp\_Line.node\_1 = i;

ui->Message\_1->addItem("请选择要需删除边的终点");

break;

}

}

}

break;

case 6: // 删除边(选择边的终点)

for(int i = 1; i <= node\_num; i++)

{

if(point[i] != temp\_Point\_1 && is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS)) // 选中

{

function\_num = 5; // 重新把控制权交给功能号5(便于重复删除“边”)

temp\_Point\_2 = point[i];

temp\_Line.node\_2 = i;

if(temp\_Line.node\_1 > temp\_Line.node\_2) // 保证线的起点的下标小于终点的下标

{

int temp = temp\_Line.node\_1;

temp\_Line.node\_1 = temp\_Line.node\_2;

temp\_Line.node\_2 = temp;

}

int side\_save\_num = side\_num;

for( int j = 1; j <= side\_num; j++ )

{

if(temp\_Line.node\_1 == line[j].node\_1 && temp\_Line.node\_2 == line[j].node\_2) // 判断图中是否有该边

{

int start = line[j].node\_1;

int end = line[j].node\_2;

dis\_matrix[start][end] = dis\_matrix[end][start] = 0; // 将距离矩阵中的该边清零

for (int k = j; k < side\_num; k++) // 把(编号大于删除边的)下标前移【除ind】

{

line[k].dis = line[k + 1].dis;

line[k].node\_1 = line[k + 1].node\_1;

line[k].node\_2 = line[k + 1].node\_2;

}

line[side\_num] = line[0];

side\_num--;

break;

}

}

if(side\_save\_num == side\_num)QMessageBox::warning(this, "警告", "图中无此边！");

else

{

ui->Message\_1->clear();

ui->Message\_1->addItem("目前有结点个数：" + QString::number(node\_num));

ui->Message\_1->addItem("目前有边的条数：" + QString::number(side\_num));

ui->Message\_1->addItem("如果还要继续删除边, 请选择下一条边的起点");

break;

}

}

}

update();

break;

case 7: // 编辑结点的标签

if(node\_num == 0)

{

QMessageBox::warning(this, "警告", "没有结点可编辑！");

}

else

{

for(int i = 1; i <= node\_num; i++)

{

if(is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS))

{

QString str = QInputDialog::getText(this, "编辑框", "输入文本(最多15个字)");

if (str != "")

point\_name[i] = str.left(15); // 左对齐

break;

}

}

}

ui->Message\_1->clear();

ui->Message\_1->addItem("如果还要继续编辑结点信息, 请选择下一个结点");

break;

case 8: // 修改边(选择边的起点)

if (side\_num == 0) QMessageBox::warning(this, "警告", "无任何路径");

else

{

for (int i = 1; i <= node\_num; i++)

{

if (is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS))

{

function\_num = 9; // 找到了修改边的起点后, 还需找到其终点. 故把控制权交给功能号9

temp\_Point\_1 = point[i];

temp\_Line.node\_1 = i;

ui->Message\_1->addItem("请选择需修改边的终点");

break;

}

}

}

break;

case 9: // 修改边(选择边的终点)

for (int i = 1; i <= node\_num; i++)

{

if (point[i] != temp\_Point\_1 && is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS)) //不能和起点重合

{

function\_num = 8; // 重新把控制权交给功能号8(便于重复修改“边”)

temp\_Point\_2 = point[i];

temp\_Line.node\_2 = i;

if (temp\_Line.node\_1 > temp\_Line.node\_2) // 确保修改边起点下标小于终点的下标

{

int temp = temp\_Line.node\_1;

temp\_Line.node\_1 = temp\_Line.node\_2;

temp\_Line.node\_2 = temp;

}

bool temp\_flag = false; // 临时标记(看是否已作修改)

for (int j = 1; j <= side\_num; j++)

{

if (temp\_Line.node\_1 == line[j].node\_1 && temp\_Line.node\_2 == line[j].node\_2) // 判断图中有无此边

{

double dis = dis\_matrix[temp\_Line.node\_1][temp\_Line.node\_2];

double number = QInputDialog::getDouble(this, "修改边的长度", "输入浮点数(0.0001~999999)", dis, 0.0001, 999999, 4);

if(number)

dis\_matrix[temp\_Line.node\_1][temp\_Line.node\_2] = dis\_matrix[temp\_Line.node\_2][temp\_Line.node\_1] = number; // 若输入了数，则边进行修改

temp\_flag = true;

break;

}

}

if (!temp\_flag) QMessageBox::warning(this, "警告", "图中无此边！");

else ui->Message\_1->addItem("如果还要继续修改边, 请选择下一条边的起点");

update();

break;

}

}

break;

}

}

}

//编辑按键

void creat\_map::**on\_add\_node\_clicked**()

{

Recover();

if(function\_num != 1)

{

function\_num = 1;

ui->add\_node->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->add\_node->setText("停止该操作");

ui->Message\_1->clear();

ui->Message\_1->addItem("请选择一个位置添加新结点");

}

else

{

function\_num = 0;

ui->add\_node->setText("添加点");

ui->Message\_1->clear();

}

}

void creat\_map::**on\_add\_edge\_clicked**()

{

Recover();

if(function\_num != 2)

{

function\_num = 2;

ui->add\_edge->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->add\_edge->setText("停止该操作");

ui->Message\_1->clear();

ui->Message\_1->addItem("请选择新增边的起点");

}

else

{

function\_num = 0;

ui->add\_edge->setText("添加边");

ui->Message\_1->clear();

}

}

void creat\_map::**on\_del\_node\_clicked**()

{

Recover();

if(function\_num != 4)

{

function\_num = 4;

ui->del\_node->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->del\_node->setText("停止该操作");

ui->Message\_1->clear();

ui->Message\_1->addItem("请选择删除结点的位置");

}

else

{

function\_num = 0;

ui->del\_node->setText("删除点");

ui->Message\_1->clear();

}

}

void creat\_map::**on\_del\_edge\_clicked**()

{

Recover();

if(function\_num != 5)

{

function\_num = 5;

ui->del\_edge->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->del\_edge->setText("停止该操作");

ui->Message\_1->clear();

ui->Message\_1->addItem("请选择需删除边的起点");

}

else

{

function\_num = 0;

ui->del\_edge->setText("删除边");

ui->Message\_1->clear();

}

}

void creat\_map::**on\_mod\_node\_clicked**()

{

Recover();

if(function\_num != 7)

{

function\_num = 7;

ui->mod\_node->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->mod\_node->setText("停止该操作");

ui->Message\_1->addItem("请选择需编辑的结点");

}

else

{

function\_num = 0;

ui->mod\_node->setText("修改点");

ui->Message\_1->clear();

}

}

void creat\_map::**on\_mod\_edge\_clicked**()

{

Recover();

if(function\_num != 8)

{

function\_num = 8;

ui->mod\_edge->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->mod\_edge->setText("停止该操作");

ui->Message\_1->addItem("请选择需要修改的边的起点");

}

else

{

function\_num = 0;

ui->mod\_edge->setText("修改边");

ui->Message\_1->clear();

}

}

//返回按键

void creat\_map::**on\_back\_clicked**()

{

MainWindow \*p=new MainWindow();

p->show();

this->close();

}

//展示边长

void creat\_map::**on\_showlen\_clicked**()

{

if(!show\_len\_flag )

{

show\_len\_flag = true;

ui->showlen->setText("隐藏所有边的长");

ui->showlen->setStyleSheet("border-image: url(../image/btn\_2.png);");

}

else

{

show\_len\_flag = false;

ui->showlen->setText("显示所有边的长"); // 每点一下切换一下状态

ui->showlen->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

update();

}

//加载地图

void creat\_map::**on\_load\_map\_clicked**()

{

Recover();

ui->load\_map->setStyleSheet("border-image: url(../image/btn\_2.png);");

using namespace std;

QString filename = QFileDialog::getOpenFileName(this, "选择文件", "", "地图文件(\*.My\_map)");

if(filename != "") // 判断文件地址是否输入成功

{

int x, y; // 用于读取征信数据

string str; // 用于读取字符串数据

ifstream in(filename.toStdString());

if(in.is\_open())

{

in >> side\_num >> node\_num;

for( int i = 1; i <= node\_num; i++ )

{

in >> x >> y; // 读取每一个点的横纵坐标

point[i].setX(x);

point[i].setY(y);

in >> str;

point\_name[i] = QString::fromStdString(str);

in >> str;

point\_info[i] = QString::fromStdString(str);

}

for( int i = 1; i <= node\_num; i++ )

for (int j = 1; j <= node\_num; j++)

in >> dis\_matrix[i][j]; // 读取距离矩阵

for( int i = 1; i <= side\_num; i++ )

in >> line[i].ind >> line[i].node\_1 >> line[i].node\_2 >> line[i].dis; // 读取每条边的数据

in >> str; // 背景图片读取路径

pic\_path = QString::fromStdString(str);

QMessageBox::information(this, "提示", "读取地图成功");

in.close(); // 关闭文件

}

else

{

QMessageBox::information(this, "提示", "读取地图失败");

}

}

ui->load\_map->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

//保存地图

void creat\_map::**on\_save\_map\_clicked**()

{

using namespace std;

Recover();

ui->save\_map->setStyleSheet("border-image: url(../image/btn\_2.png);");

QString filename = QFileDialog::getSaveFileName(this, "保存文件", "", "地图文件(\*.My\_map)"); // 文件扩展名为.My\_map

if(filename != "") // 判断文件地址是否输入成功

{

ofstream out(filename.toStdString()); // 转换为标准字符串并进行保存

if(out.is\_open())

{

out << side\_num << ' ' << node\_num << endl; // 先输入边的条数和结点个数

for( int i = 1; i <= node\_num; i++ )

out << point[i].x() << ' ' << point[i].y() << ' ' << point\_name[i].toStdString()<< ' ' << point\_info[i].toStdString() << endl;

for( int i = 1; i <= node\_num; i++ )

{

for( int j = 1; j <= node\_num; j++ )

out << dis\_matrix[i][j] << ' ';

out << endl;

}

for( int i = 1; i <= side\_num; i++ )

out << line[i].ind << ' ' << line[i].node\_1 << ' ' << line[i].node\_2 << ' ' << line[i].dis << endl;

out << pic\_path.toStdString() << endl;

QMessageBox::information(this, "提示", "保存地图成功");

out.close(); // 关闭文档

}

else

QMessageBox::information(this, "提示", "保存地图失败");

}

ui->save\_map->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

//加载背景图

void creat\_map::**on\_load\_bg\_clicked**()

{

Recover();

ui->load\_bg->setStyleSheet("border-image: url(../image/btn\_2.png);");

QString temp = QFileDialog::getOpenFileName(this, "选择文件", "", "图像文件(\*.jpg \*.jpeg \*.bmg \*.png \*.gif)"); // 设定只支持三种图片格式

if(temp != "")

pic\_path = temp; // 传入图片存储路径

update(); // 重绘(背景图片就会被画上去)

ui->load\_bg->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

//清空背景

void creat\_map::**on\_clear\_bg\_clicked**()

{

QMessageBox::StandardButton answer;

Recover();

ui->clear\_bg->setStyleSheet("border-image: url(../image/btn\_2.png);");

answer = QMessageBox::question(this, "提示", "确认要清除屏幕吗？", QMessageBox::Yes | QMessageBox::No);

if (answer == QMessageBox::Yes)

{

pic\_path = ""; // 图片路径清零

for ( int i = 1; i <= node\_num; i++ )

{

point[i] = point[0]; // 各个点的坐标位置清零

point\_name[i] = point\_name[0]; // 各个点的标签清零

for ( int j = 1; j <= node\_num; j++ )

dis\_matrix[i][j] = 0; // 距离矩阵清零

}

for (int i = 1; i <= node\_num; i++)

line[i] = line[0]; // 各条边清零

function\_num = node\_num = side\_num = info\_ind = 0; // 功能号、结点数、边数、结点标签的数组下标都清零

ui->Message\_1->clear();

}

ui->clear\_bg->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

## Main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

## Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "map\_query.h"

#include "creat\_map.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

//设计地图

void MainWindow::**on\_pushButton\_clicked**()

{

creat\_map \*p=new creat\_map();

p->show();

this->close();

}

//查询地图

void MainWindow::**on\_pushButton\_2\_clicked**()

{

map\_query \*p=new map\_query;

p->show();

this->close();

}

## Map\_query.cpp

#include "map\_query.h"

#include "ui\_map\_query.h"

#include"mainwindow.h"

#include <QtWidgets>

#include <vector>

#include <fstream>

#include <algorithm>

#include <iostream>

#include <QApplication>

#include <QLabel>

typedef struct line\_struct // 边的结构体

{

int ind; // 每条边的编号

int node\_1, node\_2; // 边的起点和终点

double dis; // 边的长度

bool flag; // 是否是当前标记为最短路径

}Line;

static const int show\_window\_x = 20; // 窗口的左上角x

static const int show\_window\_y = 40; // 窗口的左上角y

static const int show\_window\_width = 1200; // 窗口的宽

static const int show\_window\_height = 540; // 窗口的高

static const int Node\_MAX\_NUM = 100; // 设置最大结点数

static const int Side\_MAX\_NUM = 300; // 设置最大边数

static const int RADIUS = 5; // 点击半径

static int function\_num = 0; // 功能号(针对不同的操作会产生不同的效果)

static int node\_num; // 结点个数

static int side\_num; // 边的条数

static int info\_ind; // 结点标签的数组下标

static QPoint point[Node\_MAX\_NUM]; // 结点数组

static Line line[Side\_MAX\_NUM]; // 边数组

static QString point\_name[Node\_MAX\_NUM]; // 结点的编号标签数组

static QString point\_info[Node\_MAX\_NUM]; // 结点的信息标签数组

static double dis\_matrix[Node\_MAX\_NUM][Node\_MAX\_NUM]; // 距离矩阵

static QVector<int> shortest\_path; //最短路径

static QVector<QVector<int>> res; //所有路径

static QString pic\_path; // 图片路径

static bool shortest\_flag; // 两点之间的最短路径的绘制标志

//构造函数

map\_query::map\_query(QWidget \*parent) :QDialog(parent),ui(new Ui::map\_query)

{

ui->setupUi(this);

Recover();

}

//析构函数

map\_query::~map\_query()

{

delete ui;

}

//重置按键

void map\_query::Recover()

{

ui->Message\_1->clear();

ui->shortest\_rode->setText("求两点间最短路径");

ui->all\_rode->setText("求两点间所有路径");

ui->load\_map->setText("加载地图");

ui->clear->setText("清除地图");

ui->mintree->setText("最小生成树");

ui->back->setText("返回");

ui->shortest\_rode->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->all\_rode->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->load\_map->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->clear->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->mintree->setStyleSheet("border-image: url(../image/btn\_1.png);");

ui->back->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

// 计算两点间距离的函数

double map\_query::Count\_distanse(QPoint p\_1, QPoint p\_2)

{

double delta\_x, delta\_y;

delta\_x = p\_1.x() - p\_2.x();

delta\_y = p\_1.y() - p\_2.y();

return sqrt( delta\_x\*delta\_x + delta\_y\*delta\_y );

}

// 判断目标是否选中的函数

bool map\_query::is\_Click\_Suc(QPoint x, QPoint y, int n)

{

if (Count\_distanse(x, y) <= n)

return true;

else

return false;

}

// 画带箭头的线

void drawArrow(QPoint startPoint, QPoint endPoint, QPainter &p)

{

double par = 15.0; // 箭头部分三角形的腰长

double slopy = atan2((endPoint.y() - startPoint.y()), (endPoint.x() - startPoint.x()));

double cosy = cos(slopy);

double siny = sin(slopy);

p.setBrush(Qt::red); // 内部填充(白色)

QPoint point1 = QPoint(endPoint.x() + int(-par\*cosy - (par / 2.0\*siny)), endPoint.y() + int(-par\*siny + (par / 2.0\*cosy)));

QPoint point2 = QPoint(endPoint.x() + int(-par\*cosy + (par / 2.0\*siny)), endPoint.y() - int(par / 2.0\*cosy + par\*siny));

QPoint points[3] = { endPoint, point1, point2 };

p.setRenderHint(QPainter::Antialiasing, true); // 消锯齿

QPen drawTrianglePen; // 创建画笔

drawTrianglePen.setColor(Qt::red);

drawTrianglePen.setWidth(1);

p.setPen(drawTrianglePen);

p.drawPolygon(points, 3); // 绘制箭头部分

int offsetX = int(par\*siny / 3);

int offsetY = int(par\*cosy / 3);

QPoint point3, point4;

point3 = QPoint(endPoint.x() + int(-par\*cosy - (par / 2.0\*siny)) + offsetX, endPoint.y() + int(-par\*siny + (par / 2.0\*cosy)) - offsetY);

point4 = QPoint(endPoint.x() + int(-par\*cosy + (par / 2.0\*siny) - offsetX), endPoint.y() - int(par / 2.0\*cosy + par\*siny) + offsetY);

QPoint arrowBodyPoints[3] = { startPoint, point3, point4 };

p.drawPolygon(arrowBodyPoints, 3); // 绘制箭身部分

}

//绘图

void map\_query::paintEvent(QPaintEvent\*){

QPainter painter(this); // 画笔，this为当前窗口

QFont font1("ZYSong 18030", 8); // 新建 1 号字体

QFont font2("ZYSong 18030", 10); // 新建 2 号字体

this->setMaximumSize(1500,900); // 固定大小

this->setMinimumSize(1500,900);

if( pic\_path == "" ) // 如果无背景图片

painter.setBrush(Qt::white); // 白底填充

else

{

painter.drawPixmap(show\_window\_x, show\_window\_y, show\_window\_width, show\_window\_height, QPixmap(pic\_path)); // 载入背景图片并显示出来

}

painter.drawRect(show\_window\_x, show\_window\_y, show\_window\_width, show\_window\_height); // 绘制一个矩形框

painter.setFont(font2);

painter.setRenderHint(QPainter::Antialiasing, true); // 设置:绘图光滑

//划线

for( int i = 1; i <= side\_num; i++ )

{

if(!line[i].flag)

{

painter.setPen(QPen(Qt::blue,2)); // 设置画笔颜色为蓝色

painter.drawLine(point[line[i].node\_1], point[line[i].node\_2]); //两点连线

}

}

for( int i = 1; i <= side\_num; i++ )

{

if(line[i].flag)

{

painter.setPen(QPen(Qt::red,4)); // 设置画笔颜色为红色(最小生成树)

painter.drawLine(point[line[i].node\_1], point[line[i].node\_2]); //两点连线

}

}

if(shortest\_flag)

{

for( int i = shortest\_path.size()-1 ; i >0; i-- ) // 画最短路径

drawArrow(point[shortest\_path[i]], point[shortest\_path[i-1]], painter);

}

//标点

painter.setFont(font2);

for(int i = 1; i <= node\_num; i++)

{

painter.setPen(Qt::black); // 黑色

painter.setBrush(Qt::yellow); // 内部填充(黄色)

painter.drawEllipse(point[i], 4, 4); // 把点画出来

painter.drawText(QPoint(point[i].x() + 5, point[i].y() + 5), point\_name[i]);

}

}

//鼠标点击事件

void map\_query::mousePressEvent(QMouseEvent\* e){

if (e->button() == Qt::LeftButton) // 按左键

{

QPoint cur\_click\_pos = e->pos();

if(cur\_click\_pos.x() >= show\_window\_x &&cur\_click\_pos.x() <= show\_window\_x+show\_window\_width

&& cur\_click\_pos.y() >= show\_window\_y &&cur\_click\_pos.y() <= show\_window\_y+show\_window\_height)

{

for(int i = 1; i < node\_num; i++)

{

if(is\_Click\_Suc(cur\_click\_pos, point[i], RADIUS+10)) // 判断鼠标所点击位置和图上所有已添加的结点位置

{

newshowmess=new showmess;

newshowmess->senddata(point\_name[i],point\_info[i]);

newshowmess->show();

}

}

}

}

}

//最短路径

bool dijkstra(int a, int b)

{

double g[Node\_MAX\_NUM][Node\_MAX\_NUM], dist[Node\_MAX\_NUM];

bool st[Node\_MAX\_NUM] = { 0 };

int prev[Node\_MAX\_NUM];

memset(prev, -1, sizeof prev);

for (int i = 1; i <= node\_num; i++) {

dist[i] = 1000000;

for (int j = 1; j <= node\_num; j++) {

if (i == j) g[i][j] = 0;

else if (dis\_matrix[i][j] == 0) g[i][j] = 1000000;

else g[i][j] = dis\_matrix[i][j];

}

}

dist[a] = 0;

for (int i = 1; i <= node\_num; i++)

{

// 寻找距离最小的点

int t = -1;

for (int j = 1; j <= node\_num; j++) {

if (!st[j] && (t == -1 || dist[t] > dist[j])) t = j;

}

// 更新

st[t] = true;

for (int j = 1; j <= node\_num; j++)

if (dist[t] + g[t][j] < dist[j]) {

dist[j] = dist[t] + g[t][j];

prev[j] = t;

}

}

if (dist[b] == 1000000) return false;

shortest\_path.clear();

for (; b != -1; b = prev[b]) shortest\_path.push\_back(b);

return true;

}

//所有路径

void dfs(QVector<QVector<int>> &res, QVector<bool>& flag, QVector<int>& temp, int cur, int end) {

if (cur == end) {

res.push\_back(temp);

return;

}

for (int i = 1; i <= node\_num; i++) {

if (dis\_matrix[cur][i] == 0) continue;

if (!flag[i]) {

flag[i] = true;

temp.push\_back(i);

dfs(res, flag, temp, i, end);

flag[i] = false;

temp.pop\_back();

}

}

}

void allPathsSourceTarget(int a, int b) {

res.clear();

QVector<bool> flag(Node\_MAX\_NUM);

flag[a] = true;

QVector<int> temp = { a };

dfs(res, flag, temp, a, b);

}

//prim最小生成树

QVector<QPair<int,int>> prim() {

double gra[Node\_MAX\_NUM][Node\_MAX\_NUM], dis[Node\_MAX\_NUM], sum = 0;

bool flag[Node\_MAX\_NUM] = { 0 };

int prev[Node\_MAX\_NUM];

memset(prev, -1, sizeof prev);

QVector<QPair<int,int>> res;

for (int i = 1; i <= node\_num; i++) {

dis[i] = 1000000;

for (int j = 1; j <= node\_num; j++) {

if (i == j) gra[i][j] = 0;

else if (dis\_matrix[i][j] == 0) gra[i][j] = 1000000;

else gra[i][j] = dis\_matrix[i][j];

}

}

for (int i = 1; i <= node\_num; i++) {

int t = -1;

for (int j = 1; j <= node\_num; j++) {

if (!flag[j] && (t == -1 || dis[t] > dis[j])) t = j;

}

if (i != 1) sum += dis[t];

flag[t] = 1;

for (int j = 1; j <= node\_num; j++) {

if (!flag[j] && dis[j] > gra[t][j]) {

dis[j] = gra[t][j];

prev[j] = t;

}

}

}

for (int i = 1; i <= node\_num; i++) {

if (prev[i] != -1) res.push\_back({ i,prev[i] });

}

return res;

}

//按键

//返回

void map\_query::on\_back\_clicked()

{

MainWindow \*p=new MainWindow;

p->show();

this->close();

}

//加载地图

void map\_query::on\_load\_map\_clicked()

{

Recover();

ui->load\_map->setStyleSheet("border-image: url(../image/btn\_2.png);");

using namespace std;

QString filename = QFileDialog::getOpenFileName(this, "选择文件", "", "地图文件(\*.My\_map)");

if(filename != "") // 判断文件地址是否输入成功

{

int x, y; // 用于读取征信数据

string str; // 用于读取字符串数据

ifstream in(filename.toStdString());

if(in.is\_open())

{

in >> side\_num >> node\_num;

for( int i = 1; i <= node\_num; i++ )

{

in >> x >> y; // 读取每一个点的横纵坐标

point[i].setX(x);

point[i].setY(y);

in >> str;

point\_name[i] = QString::fromStdString(str);

in >> str;

point\_info[i] = QString::fromStdString(str);

}

for( int i = 1; i <= node\_num; i++ )

for (int j = 1; j <= node\_num; j++)

in >> dis\_matrix[i][j]; // 读取距离矩阵

for( int i = 1; i <= side\_num; i++ )

in >> line[i].ind >> line[i].node\_1 >> line[i].node\_2 >> line[i].dis; // 读取每条边的数据

in >> str; // 背景图片读取路径

pic\_path = QString::fromStdString(str);

QMessageBox::information(this, "提示", "读取地图成功");

in.close(); // 关闭文件

//填写选择框

ui->start->clear();

ui->end->clear();

for( int i = 1; i <= node\_num; i++ ){

ui->start->addItem(point\_name[i]);

ui->end->addItem(point\_name[i]);

}

}

else

{

QMessageBox::information(this, "提示", "读取地图失败");

}

}

ui->load\_map->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

//清除地图

void map\_query::on\_clear\_clicked()

{

QMessageBox::StandardButton answer;

Recover();

ui->clear->setStyleSheet("border-image: url(../image/btn\_2.png);");

answer = QMessageBox::question(this, "提示", "确认要清除屏幕吗？", QMessageBox::Yes | QMessageBox::No);

if (answer == QMessageBox::Yes)

{

pic\_path = ""; // 图片路径清零

for ( int i = 1; i <= node\_num; i++ )

{

point[i] = point[0]; // 各个点的坐标位置清零

point\_name[i] = point\_name[0]; // 各个点的标签清零

for ( int j = 1; j <= node\_num; j++ )

dis\_matrix[i][j] = 0; // 距离矩阵清零

}

for (int i = 1; i <= node\_num; i++)

line[i] = line[0]; // 各条边清零

function\_num = node\_num = side\_num = info\_ind = 0; // 功能号、结点数、边数、结点标签的数组下标都清零

ui->Message\_1->clear();

}

ui->clear->setStyleSheet("border-image: url(../image/btn\_1.png);");

}

//最短路径

void map\_query::on\_shortest\_rode\_clicked()

{

if(function\_num != 1)

{

Recover();

function\_num = 1;

ui->shortest\_rode->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->shortest\_rode->setText("停止该操作");

QString s1=ui->start->currentText();

QString s2=ui->end->currentText();

int a,b;

for(int i=1;i<node\_num;i++){

if(point\_name[i]==s1)a=i;

if(point\_name[i]==s2)b=i;

}

if(dijkstra(a,b)){

shortest\_flag=true;

update();

}

else {

QMessageBox::warning(this, "警告", "这两点间没有通路");

function\_num = 0;

Recover();

}

}

else

{

Recover();

function\_num = 0;

shortest\_flag=false;

ui->Message\_1->clear();

update();

}

}

//所有路径

void map\_query::on\_all\_rode\_clicked()

{

Recover();

if(function\_num != 2)

{

function\_num = 2;

ui->all\_rode->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->all\_rode->setText("停止该操作");

ui->Message\_1->clear();

QString s1=ui->start->currentText();

QString s2=ui->end->currentText();

int a,b;

for(int i=1;i<node\_num;i++){

if(point\_name[i]==s1)a=i;

if(point\_name[i]==s2)b=i;

}

allPathsSourceTarget(a,b);

QString tmp;

for(int i=0;i<res.size();i++){

for(int j=0;j<res[i].size();j++){

tmp+=point\_name[res[i][j]];

if(j!=res[i].size()-1)tmp+="->";

}

ui->Message\_1->addItem(tmp);

tmp.clear();

}

}

else

{

function\_num = 0;

ui->all\_rode->setText("求两点间所有路径");

ui->Message\_1->clear();

}

}

//最小生成树

void map\_query::on\_mintree\_clicked()

{

Recover();

if(function\_num != 1)

{

function\_num = 1;

ui->mintree->setStyleSheet("border-image: url(../image/btn\_2.png);");

ui->mintree->setText("停止该操作");

ui->Message\_1->clear();

QVector<QPair<int,int>>ans=prim();

for(int i=0;i<ans.size();i++){

for(int j=1;j<=side\_num;j++){

if((line[j].node\_1==ans[i].first&&line[j].node\_2==ans[i].second)||

(line[j].node\_2==ans[i].first&&line[j].node\_1==ans[i].second)){

line[j].flag=true;

break;

}

}

}

update();

}

else

{

function\_num = 0;

ui->mintree->setText("求最小生成树");

ui->Message\_1->clear();

for(int j=1;j<=side\_num;j++)line[j].flag=false;

update();

}

}

## Showmess.cpp

#include "showmess.h"

#include "ui\_showmess.h"

showmess::showmess(QWidget \*parent) :

QDialog(parent),

ui(new Ui::showmess)

{

ui->setupUi(this);

}

showmess::~showmess()

{

delete ui;

}

void showmess::senddata(QString s1,QString s2){

ui->title->setText(s1);

ui->content->setText(s2);

}

## Creat\_map.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>creat\_map</class>

<widget class="QDialog" name="creat\_map">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1500</width>

<height>900</height>

</rect>

</property>

<property name="minimumSize">

<size>

<width>1500</width>

<height>900</height>

</size>

</property>

<property name="maximumSize">

<size>

<width>1500</width>

<height>900</height>

</size>

</property>

<property name="windowTitle">

<string>Dialog</string>

</property>

<widget class="QListWidget" name="Message\_1">

<property name="geometry">

<rect>

<x>20</x>

<y>650</y>

<width>1461</width>

<height>141</height>

</rect>

</property>

</widget>

<widget class="QWidget" name="layoutWidget">

<property name="geometry">

<rect>

<x>1280</x>

<y>20</y>

<width>201</width>

<height>621</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QPushButton" name="showlen">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>显示路径长度</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="load\_map">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>加载地图</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="save\_map">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>保存地图</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="load\_bg">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>加载背景</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="clear\_bg">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>清除屏幕</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="back">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>返回</string>

</property>

</widget>

</item>

</layout>

</widget>

<widget class="QWidget" name="layoutWidget">

<property name="geometry">

<rect>

<x>20</x>

<y>800</y>

<width>1461</width>

<height>81</height>

</rect>

</property>

<layout class="QHBoxLayout" name="horizontalLayout">

<property name="spacing">

<number>12</number>

</property>

<item>

<widget class="QPushButton" name="add\_node">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>添加点</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="add\_edge">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>添加边</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="del\_node">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>删除点</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="del\_edge">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>删除边</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="mod\_node">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>修改点</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="mod\_edge">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>修改边</string>

</property>

</widget>

</item>

</layout>

</widget>

</widget>

<resources/>

<connections/>

</ui>

## Mainwindow.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1284</width>

<height>668</height>

</rect>

</property>

<property name="windowTitle">

<string>MainWindow</string>

</property>

<widget class="QWidget" name="centralwidget">

<widget class="QPushButton" name="pushButton">

<property name="geometry">

<rect>

<x>260</x>

<y>410</y>

<width>261</width>

<height>91</height>

</rect>

</property>

<property name="font">

<font>

<pointsize>18</pointsize>

</font>

</property>

<property name="text">

<string>设计地图</string>

</property>

</widget>

<widget class="QPushButton" name="pushButton\_2">

<property name="geometry">

<rect>

<x>650</x>

<y>410</y>

<width>261</width>

<height>91</height>

</rect>

</property>

<property name="font">

<font>

<pointsize>18</pointsize>

</font>

</property>

<property name="text">

<string>查询地图</string>

</property>

</widget>

<widget class="QLabel" name="label">

<property name="geometry">

<rect>

<x>380</x>

<y>120</y>

<width>491</width>

<height>121</height>

</rect>

</property>

<property name="font">

<font>

<pointsize>18</pointsize>

</font>

</property>

<property name="text">

<string>校园景点规划和导游的系统设计</string>

</property>

</widget>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1284</width>

<height>26</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<resources/>

<connections/>

</ui>

## Map\_query.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>map\_query</class>

<widget class="QDialog" name="map\_query">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1500</width>

<height>900</height>

</rect>

</property>

<property name="minimumSize">

<size>

<width>1500</width>

<height>900</height>

</size>

</property>

<property name="maximumSize">

<size>

<width>1500</width>

<height>900</height>

</size>

</property>

<property name="windowTitle">

<string>Dialog</string>

</property>

<widget class="QListWidget" name="Message\_1">

<property name="geometry">

<rect>

<x>10</x>

<y>690</y>

<width>1451</width>

<height>191</height>

</rect>

</property>

</widget>

<widget class="QWidget" name="layoutWidget">

<property name="geometry">

<rect>

<x>10</x>

<y>600</y>

<width>1451</width>

<height>72</height>

</rect>

</property>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<widget class="QLabel" name="label">

<property name="minimumSize">

<size>

<width>100</width>

<height>40</height>

</size>

</property>

<property name="font">

<font>

<pointsize>18</pointsize>

</font>

</property>

<property name="text">

<string>起点</string>

</property>

<property name="alignment">

<set>Qt::AlignCenter</set>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="start">

<property name="minimumSize">

<size>

<width>270</width>

<height>70</height>

</size>

</property>

</widget>

</item>

<item>

<widget class="QLabel" name="label\_2">

<property name="minimumSize">

<size>

<width>100</width>

<height>40</height>

</size>

</property>

<property name="font">

<font>

<pointsize>18</pointsize>

</font>

</property>

<property name="text">

<string>终点</string>

</property>

<property name="alignment">

<set>Qt::AlignCenter</set>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="end">

<property name="minimumSize">

<size>

<width>270</width>

<height>70</height>

</size>

</property>

</widget>

</item>

</layout>

</widget>

<widget class="QWidget" name="layoutWidget">

<property name="geometry">

<rect>

<x>1240</x>

<y>10</y>

<width>221</width>

<height>571</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<property name="spacing">

<number>20</number>

</property>

<item>

<widget class="QPushButton" name="load\_map">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>加载地图</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="clear">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>清除地图</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="shortest\_rode">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>求两点间最短路径</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="all\_rode">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>求两点间所有路径</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="mintree">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>最小生成树</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="back">

<property name="minimumSize">

<size>

<width>150</width>

<height>70</height>

</size>

</property>

<property name="font">

<font>

<pointsize>12</pointsize>

</font>

</property>

<property name="text">

<string>返回</string>

</property>

</widget>

</item>

</layout>

</widget>

</widget>

<resources/>

<connections/>

</ui>

## Show\_mess.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>showmess</class>

<widget class="QDialog" name="showmess">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>699</width>

<height>435</height>

</rect>

</property>

<property name="windowTitle">

<string>Dialog</string>

</property>

<widget class="QDialogButtonBox" name="buttonBox">

<property name="geometry">

<rect>

<x>320</x>

<y>380</y>

<width>341</width>

<height>32</height>

</rect>

</property>

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

<property name="standardButtons">

<set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>

</property>

</widget>

<widget class="QTextBrowser" name="content">

<property name="geometry">

<rect>

<x>50</x>

<y>130</y>

<width>581</width>

<height>221</height>

</rect>

</property>

</widget>

<widget class="QTextBrowser" name="title">

<property name="geometry">

<rect>

<x>190</x>

<y>30</y>

<width>261</width>

<height>41</height>

</rect>

</property>

</widget>

</widget>

<resources/>

<connections>

<connection>

<sender>buttonBox</sender>

<signal>accepted()</signal>

<receiver>showmess</receiver>

<slot>accept()</slot>

<hints>

<hint type="sourcelabel">

<x>248</x>

<y>254</y>

</hint>

<hint type="destinationlabel">

<x>157</x>

<y>274</y>

</hint>

</hints>

</connection>

<connection>

<sender>buttonBox</sender>

<signal>rejected()</signal>

<receiver>showmess</receiver>

<slot>reject()</slot>

<hints>

<hint type="sourcelabel">

<x>316</x>

<y>260</y>

</hint>

<hint type="destinationlabel">

<x>286</x>

<y>274</y>

</hint>

</hints>

</connection>

</connections>

</ui>