《数据结构》课程设计总结

学号：2252941

姓名：杨瑞灵

专业：计算机科学与技术

2024 年 8 月

目　录

第一部分 算法实现设计说明.....1
1.1 题目.....
1.2 软件功能.....
1.3 设计思想.....
1.4 逻辑结构与物理结构.....
1.5 开发平台.....
1.6 系统的运行结果分析说明.....
1.7 操作说明.....
第二部分 综合应用设计说明.....
2.1 题目.....
2.2 软件功能.....
2.3 设计思想.....
2.4 逻辑结构与物理结构.....
2.5 开发平台.....
2.6 系统的运行结果分析说明.....
2.7 操作说明.....
第三部分 实践总结.....
3.1.所做的工作.....
3.2.总结与收获.....
第四部分 参考文献.....

第一部分 算法实现设计说明

1.1 题目

题号：1

分别以单链表、循环链表、双向链表为例，实现线性表的建立、插入、删除、查找等基本操作。 要求：能够把建立、插入、删除等基本操作的过程随时显示输出来。

1.2 软件功能

1. **功能描述：**

链表类型选择：用户可以选择使用单链表、循环链表或双向链表。

链表操作选择：

建立链表：初始化一个空链表。

插入元素：用户可以指定位置或元素值，在链表中插入新元素。

删除元素：用户可以指定位置或元素值来删除元素。

查找元素：用户可以指定值或位置，查找链表中的某个元素。

显示链表：实时显示链表的结构和数据内容。当链表发生变化时，界面应自动刷新，更新链表显示。

错误处理：对无效操作（如插入越界、删除不存在元素等）提供错误提示。

2. **软件界面设计**

链表类型选择区域：提供单链表、循环链表和双向链表的选择按钮。

链表选择操作区域：包含插入、删除、查找等操作按钮。

输入框：用户输入操作相关的值。

显示当前链表状态的区域：链表元素按顺序排列，实时更新。

1.3 设计思想

1.3.1 实现思路

该软件主要分为前端用户界面和后端数据处理两部分，通过用户界面实现链表的基本操作，后台数据结构负责存储和操作链表中的数据。程序运行时，用户可以通过界面选择使用的链表类型，并执行相关操作如插入、删除、查找等，软件会实时更新显示链表的结构和操作过程。

前端设计：使用 Qt 设计界面，动态更新链表结构的显示，通过信号和槽机制实现按钮

点击后的操作反馈。包括链表选择、插入、删除和查找等操作的按钮和输入框。界面上还提供链表结构的可视化显示和操作过程的文本输出。

后端设计：后端实现单链表、循环链表和双向链表的数据结构，通过类的封装实现对链表的创建、插入、删除、查找等操作。

1.3.2 算法设计

1. **链表创建**：

   初始化链表节点，将头节点指针设置为 nullptr

   循环链表，将最后一个节点指向头节点

   双向链表，一个正循环一个反循环链，同时有一个头指针和一个尾指针

2. **插入算法**：

   遍历链表，找到目标位置，将新节点插入链表，并调整前后节点的指针

3. **删除算法**：

   按值删除：遍历链表，找到与值匹配的节点，删除该节点

4. **查找算法**：

   遍历链表，找到第一个与值匹配的节点，返回其位置

5. **显示链表结构**：

   遍历链表，将每个节点的值和连接关系显示在图形界面上

   更新链表结构后，自动刷新显示

1.4 逻辑结构与物理结构

1.4.1 逻辑结构

1. **链表类（List）**：抽象出链表的基本操作，包含单链表、循环链表和双向链表的创建、插入、删除、查找等操作。

   • Node 类：

```
struct Node
{
    int data;
    Node *next;
    Node *prev;
    Node(int val) : data(val), next(nullptr), prev(nullptr) {}
};
```

   • SingleLinkedList 类：实现单链表的相关操作。

```
class SinglyLinkedList
```

```
{
public:
    SinglyLinkedList() : head(nullptr), len(0) {}
    // 建立
    void clear();
    // 头部插入
    void insert(int);
    // 删除指定
    bool remove(int);
    // 寻找第一个指定值为 val 的节点?
    Node *find(int, int&);
    // 显示列表??
    string single_display();
    string circular_display();
    string double_display();
private:
    Node *head;
    int len;
};
```

- **CircularLinkedList 类**：实现循环链表的相关操作。
- **DoublyLinkedList 类**：实现双向链表的相关操作。

2. **GUI 类**：负责界面的布局和交互。
   - **Dialog 类**：负责管理整个对话的布局，包括按钮、输入框、链表显示区域等。

1.4.2 物理结构

程序结构按模块划分为多个源文件和头文件：

main.cpp：程序的入口文件，初始化应用程序，加载界面。

dialog.cpp 和 **dialog.h**：负责对话界面的实现，包括界面元素的布局和用户交互。

linkedlist.h 和 **linkedlist.cpp**：链表操作的实现，包括单链表、循环链表、双向链表的基本操作。

ui_dialog.h：由 Qt Creator 自动生成的界面文件，用于定义界面布局和控件属性。

1.5 开发平台

1.5.1 开发工具

操作系统 ：Windows 11 家庭版

开发语言 ：C++（C++11 标准以上）

开发框架 ：QT

集成开发环境：Qt Version 6.7.2

编译器 ：MinGW 7.3.2 64-bit

1.5.2 第三方库

Qt Framework：提供图形界面、信号与槽机制、事件处理等功能。

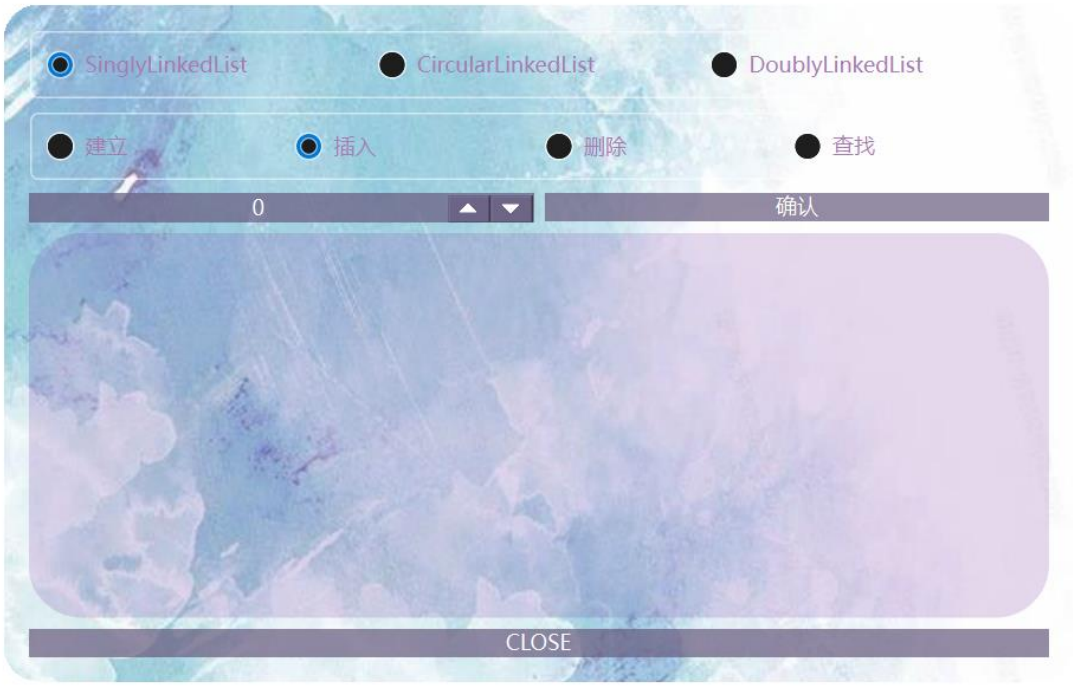
QMouseEvent：处理鼠标事件

1.5.3 运行环境

操作系统：支持 Windows、macOS、Linux 等主流操作系统。

依赖库：需要安装 Qt 框架库才能运行程序。

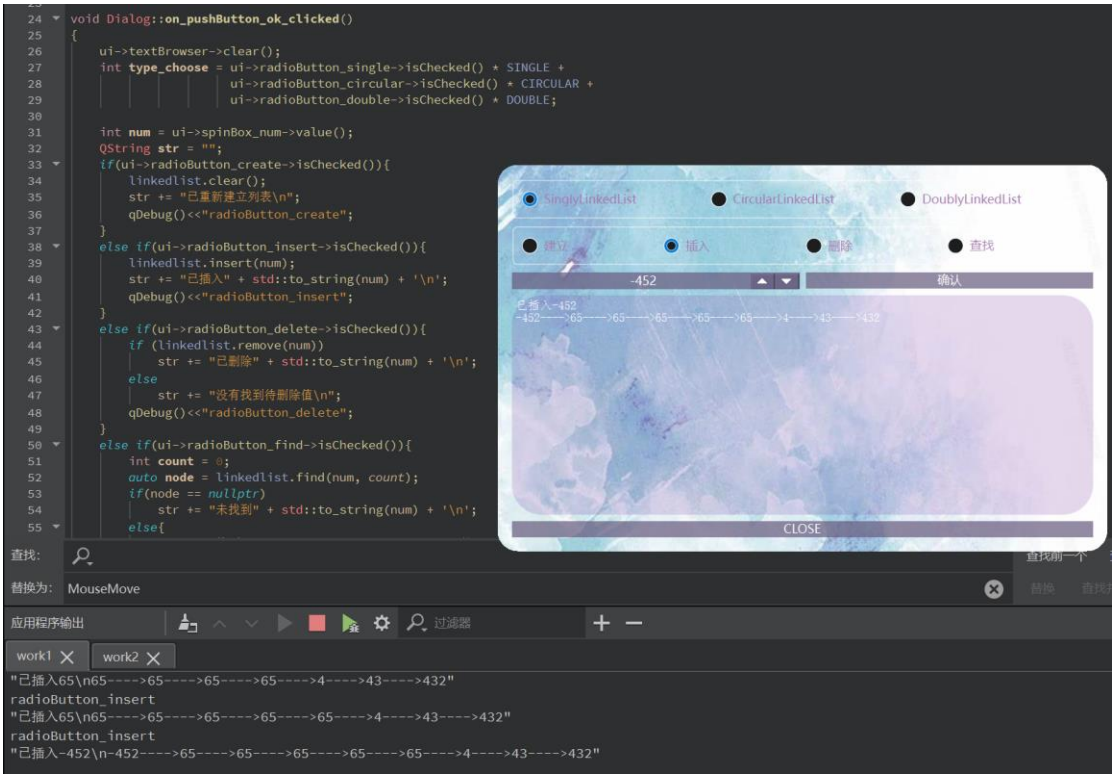
1.6 系统的运行结果分析说明



1.6.1 调试与开发过程

1. **通过 qDebug 进行输出**

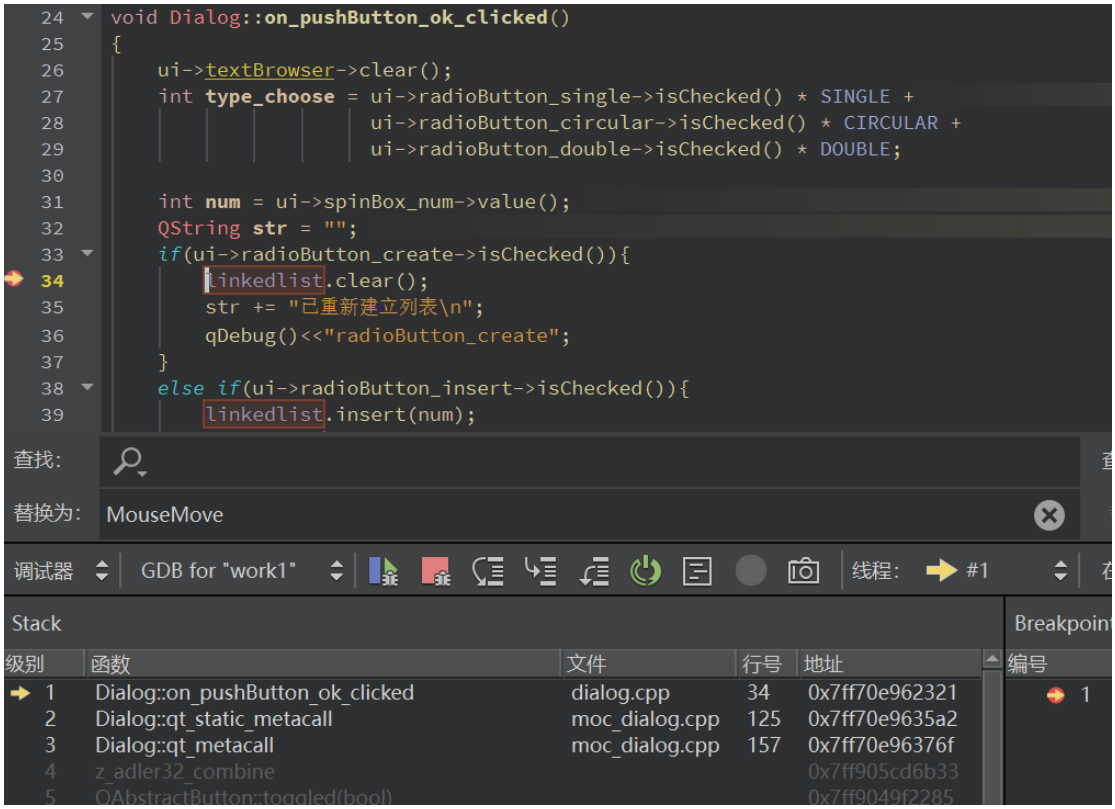
在开发过程中，使用了 Qt 框架提供的 QDebug 输出来实时输出调试信息。这些输出信息包括了程序的执行流程、变量的值、以及函数的调用。通过 QDebug 输出，可以快速定位和解决潜在的问题，如内存泄漏、逻辑错误等。这些输出信息有助于我在开发过程中跟踪代码的执行路径，并确认算法的正确性。



如上图所示，通过 qDebug 可以在程序运行期间在控制台打印所需要的调试信息，辅助我进行问题的排查与定位，更快速的实现程序的调试与逻辑的改进。

2. 通过 Qt 的调试模式进行调试分析

Qt 提供了强大的调试工具，包括调试器和分析工具。在开发过程中，使用 Qt 的调试 模式来单步执行代码、观察变量值和查看函数调用堆栈。这有助于深入了解程序的运行状态，以及在出现问题时快速定位错误所在。通过调试模式，可以逐步验证算法的正确性，并进行必要的修复和优化。



如上图所示，通过 Qt 的调试器，可以进行单步调试。

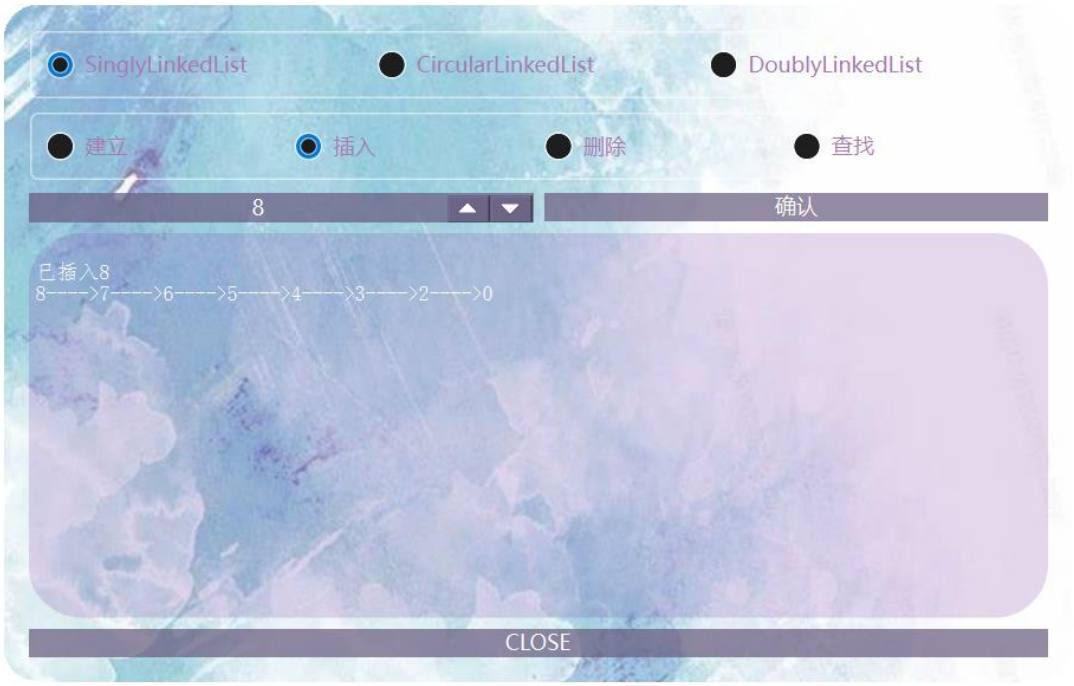
1.6.2 软件正确性、稳定性与容错能力

1. **软件正确性：**

• **测试与验证**：软件经过多次测试，确保在各种情况下都能按照预期执行。测试涵盖了单链表、循环链表和双向链表的各种基本操作，如节点的插入、删除和查找。以下是一些测试案例：

• **单链表测试**：

插入操作：向单链表中插入多个节点，并验证链表中的节点顺序正确。

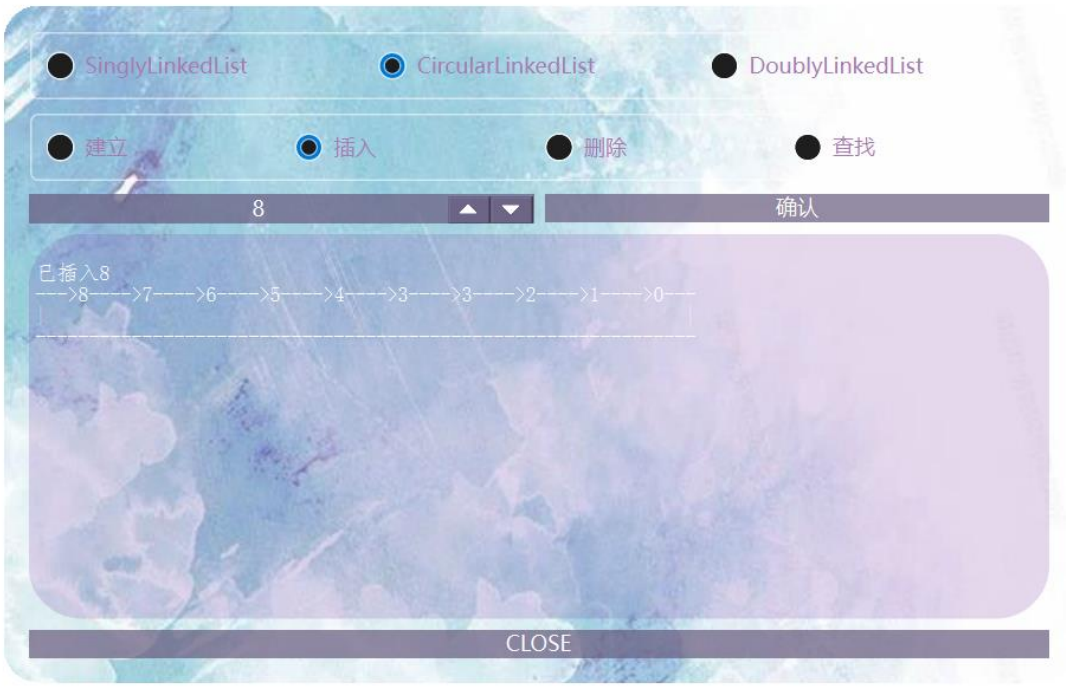


删除操作：尝试删除链表头节点、中间节点和尾节点，确保链表结构保持完整。

查找操作：查找指定节点，并验证返回的节点位置正确。

• 循环链表测试：

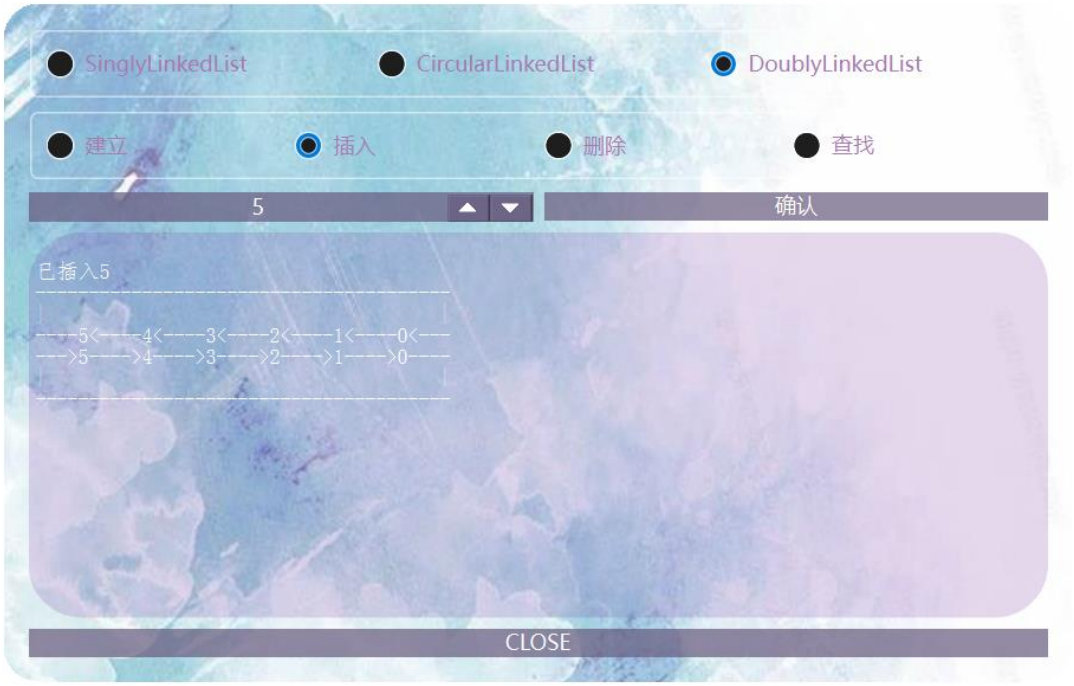
插入操作：验证循环链表中的节点能够正确地循环连接，确保最后一个节点的 next 指向头节点。



删除操作：测试删除节点后，链表仍保持循环特性，并验证链表长度正确。

• **双向链表测试：**

插入操作：验证节点在双向链表中正确插入，确保 prev 和 next 指针正确连接。



删除操作：测试删除节点后，双向链表中的节点连接依旧正确。

1. **软件稳定性：**

• **内存管理**：在链表操作过程中，软件对内存管理进行了严格控制，确保没有内存泄漏和非法访问。每次动态分配内存时，都在适当时机进行释放。例如：节点删除：在删除节点时，先断开节点的连接，再释放内存，防止出现悬空指针。

```
bool SinglyLinkedList::remove(int val)
{
    Node *current = head;
    Node *prev = nullptr;
    int count = 1;
    while (current != nullptr && current->data != val){
        prev = current;
        current = current->next;
        ++count;
    }
    // 没有找到
    if (current == nullptr){
        return false;
    }
    if (prev == nullptr){
        head = head->next;
```

```
        len--;
    }
    else{
        prev->next = current->next;
        len--;
    }
    delete current;
    return true;
}
```

• **对象生命周期管理**：通过在适当位置使用构造函数和析构函数，确保对象生命周期管理得当。例如：链表清除操作：调用 clear()方法时，确保链表中的所有节点都被正确释放，防止内存泄漏。

• **异常处理**：在链表操作中增加了异常处理机制，如当插入、删除或查找节点时出现异常情况时，能够有效捕获并进行处理，防止程序崩溃。例如：非法节点操作：在尝试操作空链表或删除不存在节点时，给出合理的提示并退出操作，确保程序稳定运行。

```
else if (ui->radioButton_delete->isChecked())
{
    if (linkedlist.remove(num))
        str += "\n 已删除" + std::to_string(num) + '\n';
    else
        str += "\n 没有找到待删除值\n";
    qDebug() << "radioButton_delete";
}
```

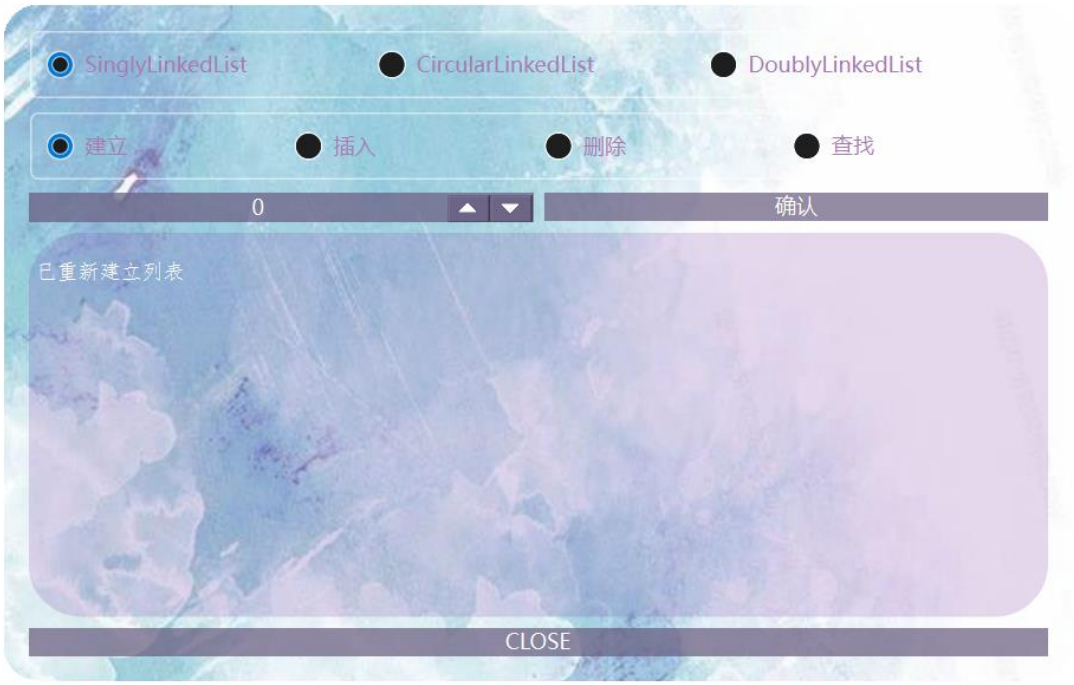
2. **软件容错能力：**

在执行插入、删除、查找等操作时，软件对输入数据进行了验证，确保操作安全。例如：删除节点：在删除操作中，软件会验证链表中是否存在该节点，如不存在则输出提示信息，并停止操作。查找节点：在查找节点时，若链表为空，软件会直接返回空指针，防止访问非法内存。

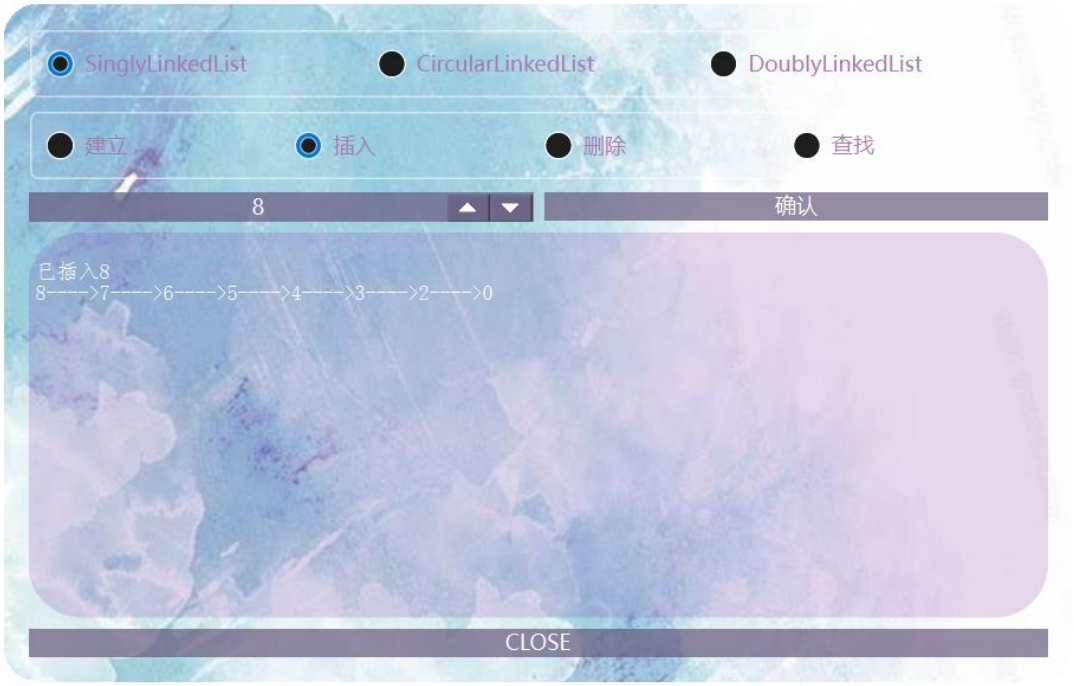
1.6.3 运行案例

下面以单链表为例，对建立、插入、删除、查找操作进行展示。

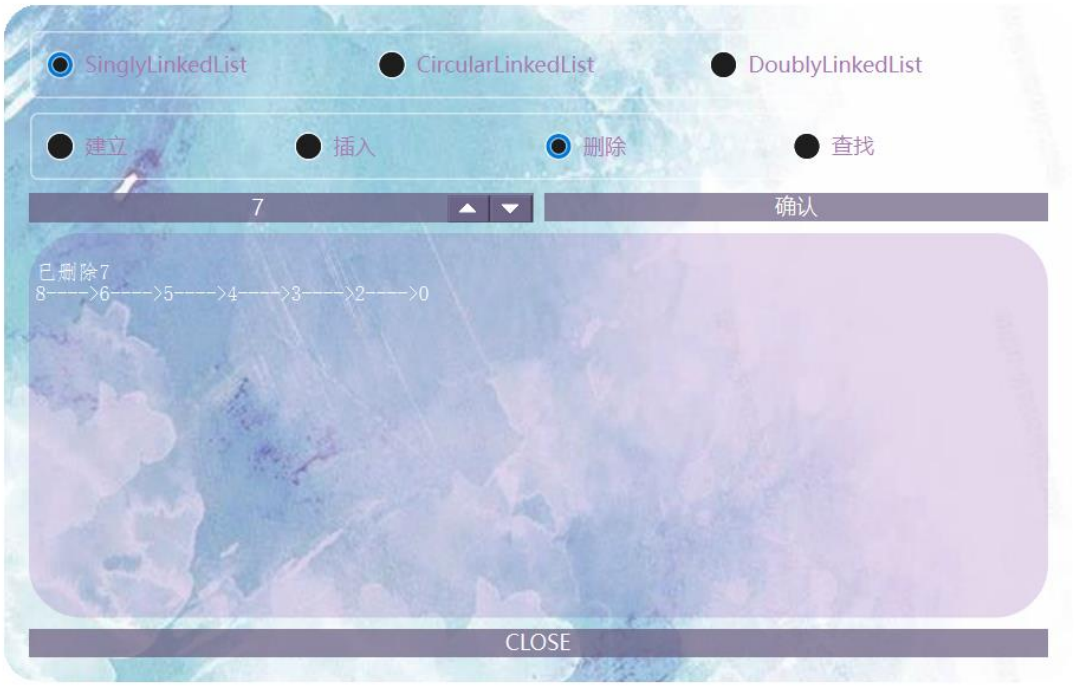
1. **单链表建立：**

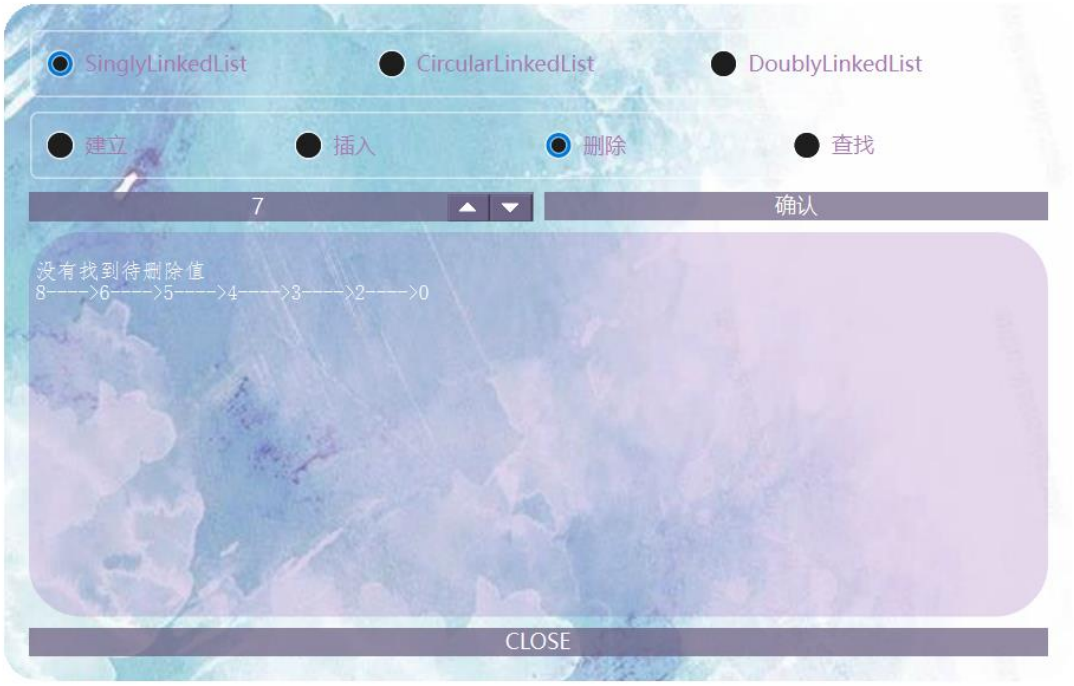


2. **插入操作**：向单链表中插入多个节点，并验证链表中的节点顺序正确。

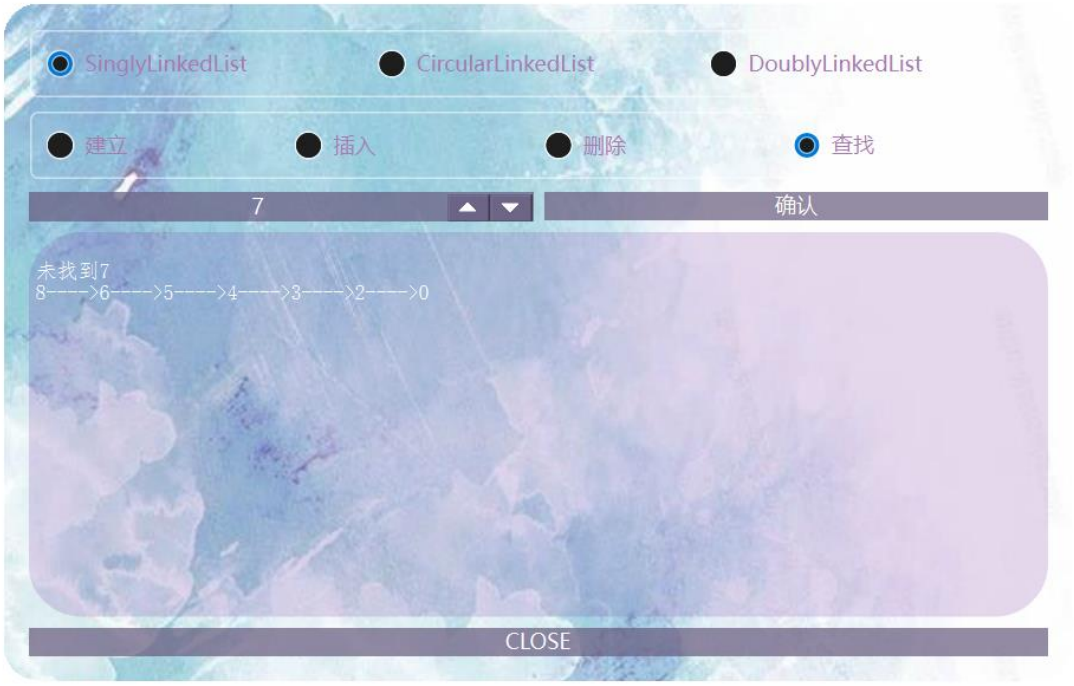


3. **删除操作**：尝试删除链表头节点、中间节点和尾节点，确保链表结构保持完整。





4. **查找操作**：查找指定节点，并验证返回的节点位置正确。



1.7 操作说明

更加所编写的软件功能，通过图文结合的方式说明基本操作过程。

第一行选择单链表、循环链表、双向链表

第二行选择具体操作建立、插入、删除、查找

第三行 spinBox 为插入、删除、查找的输入值，只能为-999 — 999 的整数，pushbutton 为确认按钮

第四行 textBrowser 用来显示具体输出值和链表可视化

第五行 CLOSE 关闭对话

由于没有 title 行，还实现了长按左键的鼠标拖动

第二部分 综合应用设计说明

2.1 题目

上海的地铁交通网路已基本成型，建成的地铁线十多条，站点上百个，现需建立

一个换乘指南打印系统，通过输入起点站和终点站，打印出地铁换乘指南，指南内容包括起点站、换乘站、终点站。

（1）图形化显示地铁网络结构，能动态添加地铁线路和地铁站点。

（2）根据输入起点站和终点站，显示地铁换乘指南。

（3）通过图形界面显示乘车路径。

2.2 软件功能

2.2.1 地铁信息查看功能

这个功能允许用户在图形界面上可视化查看上海地铁的线路，并能够通过鼠标拖动和滚轮来缩放和移动地铁网络地图。同时，用户还可以将鼠标悬浮在地铁站点上，以查看有关该

站点的详细信息，如站点名称、位置等。

• 用户启动程序后，会看到一个包含上海地铁线路的可视化地铁网络地图。

• 地铁线路以不同颜色或样式表示，各站点沿线显示。

• 用户可以通过鼠标拖动地图来移动视图，以便查看不同区域的线路。

• 用户还可以使用鼠标滚轮来缩放地图，以查看线路的细节。

• 当用户将鼠标悬浮在地铁站点上时，会弹出一个信息窗口，显示站点的相关信息， 如站点名称、位置坐标、线路信息等。

2.2.2 地铁网络编辑功能

用户可以通过图形界面动态添加地铁线路和站点，以维护地铁网络的数据。

• 点击“添加路线”，在对话框中输入新路线的名称以及颜色。

• 点击“添加站点”，在弹出的对话框中输入新站点的信息，括经纬度和名称，点击确定后，会自动刷新画布，显示新站点。

• 点击“添加连线”，在对话框中输入两个站点以及所属路线，创建已有站点连接的地铁线路关系。

2.2.2 换乘指南生成功能

根据用户输入的起点站和终点站，生成地铁换乘指南，指南内容包括起点站、换乘站、终点站。

• 用户在界面上输入起点站和终点站。

• 用户点击“确定”按钮后，程序会计算并显示地铁换乘指南。

2.2.4 图形界面显示乘车路径

在图形界面上显示用户的乘车路径，以便用户更直观地了解从起点站到终点站的路径。

• 用户在生成地铁换乘指南后，可以在图形界面上看到完整的乘车路径，包括沿途的站点和线路。

• 乘车路径上的站点和线路加粗来表示。

2.3 设计思想

2.3.1. 实现思路

1. 图形界面

• ui_mainwindow.h　　class MainWindow : public QMainWindow

主页面，包括画板显示地铁线路图，起点终点和选择按键，以及显示换成线路图的输出框

• ui_subwaycontrolwindow.h　　class SubwayControlWindow : public QWidget

添加线路、站点、连线的页面

2. 后端

• class Station　地铁站类，同时记录连线

名字、经纬度、转换成的地图上坐标、所有与此站点直接相连的边，这个站点参与的路线

```
class  Station
{
protected:
    QString name;
    double longi, lati;         // 经纬度
    QPointF coord;              // 图上的坐标位置
    QMap<QString, Edge> edges;  // 边：记录所有和 name 直接相连的 station 以及他们之间的距离
    QSet<QString> lines;        // 线路

public:
    Station();
    // 经纬度转地图位置
    void latilongi2coord();
    // 添加一条从 this 到 sta 的位于 line_name 线上的边
    State add_edge(const Station&,const QString,const float = -1);
    // 获取所有属于的线路线路
    QString get_belonglines_text();
    };
```

• class Line　路线类

名字、颜色、总站数、地点站和终点站、站点集合

```
class Line
{
protected:
    QString name;
    QColor color;
    QList<int> total_stations;            //总站数
    QList<QString> start_stas,end_stas;    //起点站和终点站
    QList<QList<QString>> sta_list;        //站点集合,避免一条线有多个分支
};
```

• class SubwaySystem　地铁线路类

站点集合、线路集合

```
class SubwaySystem
```

```
{
protected:
    QMap<QString,Station> stations;     //所有站点的集合
    QMap<QString,Line> lines;           //所有线路集合

public:
    SubwaySystem();

    State readSubwayFile(QString);
    void statisticEdges();

    State addStation(QString,double longi,double lati,QSet<QString>);
    State addLine(QString,QColor);
    State addEdge(QString,QString,QString);
    QList<QString> shortTimePath(const QString,const QString);
    QList<QString> getSameLineABPath(const QString&,const QString&)const;
};
```

• class StationQGraphicsView : public QGraphicsView 画图类

放大、缩小、重置

```
class StationQGraphicsView : public QGraphicsView
{
public:
    StationQGraphicsView(QWidget *parent = nullptr);

    void zoom_in();
    void zoom_out();
    void reset_zoom();
    void refresh();
private:
    void zoom(double);
};
```

2.3.2. 数据结构

如 2.3.1

• Class Station 记录每个站点,用 Qset 记录 station 所在线路,用 QMap 记录 station 所在连线。

由于站点、线路是名称和实体对象之间进行一一对应，故使用 Qt 中的 QMap。QMap 类似 c++中的 map,是基于红黑树实现的一种键值相对应的类型,使用效果非常类似于哈希表，但是其查找的复杂度为 O(logn)，稍慢于哈希表。程序中由于地铁站点和线路数目有限，故使用 QMap 也可以达到很高的效率。 在地铁站点所属线路等方面，我还使用了集合这一数据

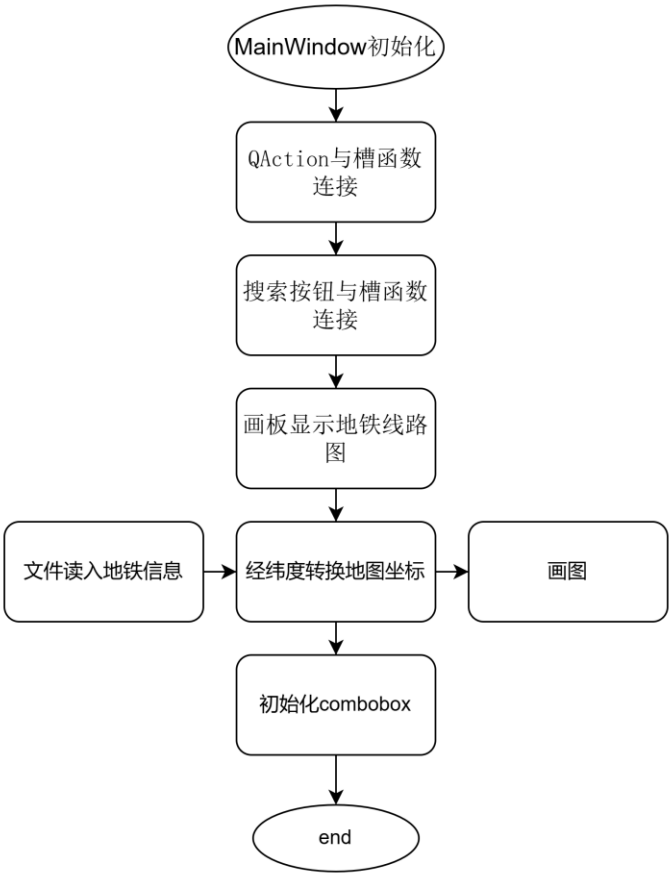
结构。Qt 中的集合类是 QSet， 集合的特性使得元素不能重复插入集合中，非常适合某些特殊场合的要求。

• **Class Line** 记录每条线路，用 QList<QList<QString>>记录站点，注意可能有岔路，及一个起点多个终点的情况，所以套两层 QList

• **Class SubwaySystem** 记录整个地铁线路，QMap<QString,Station>记录所有站点，QMap<QString,Line>记录所有线路

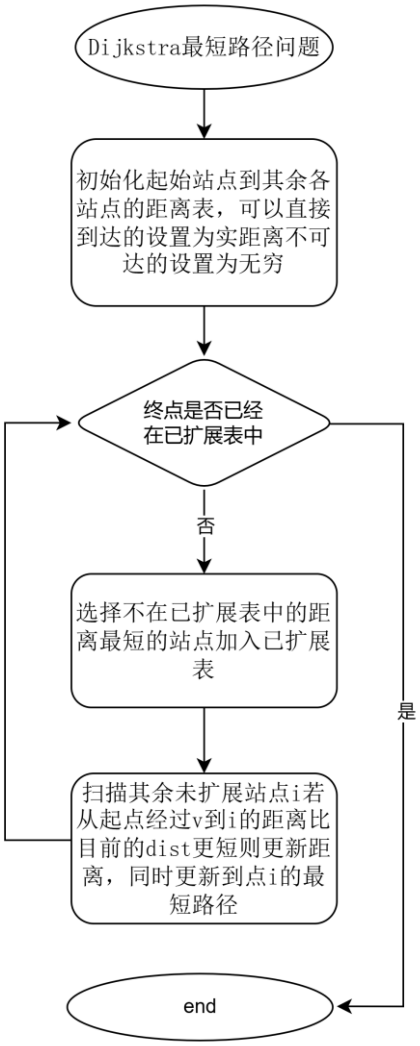
2.3.3.算法设计基本流程

1. **MainWindow 初始化：**




```
connect(ui->action_addstation, &QAction::triggered, this, &MainWindow::add_station);
connect(ui->action_addedge, &QAction::triggered, this, &MainWindow::add_edge);
connect(ui->action_addline, &QAction::triggered, this, &MainWindow::add_line);
connect(ui->action_subway, &QAction::triggered, this,
&MainWindow::draw_subway_system);
connect(ui->action_zoomin, &QAction::triggered,
ui->graphicsView,&StationQGraphicsView::zoom_in);
connect(ui->action_zoomout, &QAction::triggered,
ui->graphicsView,&StationQGraphicsView::zoom_out);
```

2. **查找路径**：使用了 Dijkstra 求解最短路径问题，并且显示在画板上



3. 加入线路站点和连线：

如上 1.中槽函数 add_station、add_line、add_edge，以 add_line 为例：

```
void MainWindow::add_line()
{
    SubwayControlWindow *sub_window = new SubwayControlWindow(1,&this->subsys);
    sub_window->show();
}
```

一旦触发此槽函数，就会新建 SubwayControlWindow 类，显示图形化界面，并调用 init 函数

```
void SubwayControlWindow::init_tab_line()
{
    connect(ui->pushButton_color,&QPushButton::clicked,this,&SubwayControlWindow::get_color);
    connect(ui->pushButton_lineok,&QPushButton::clicked,this,&SubwayControlWindow::submit_tab_line);
    connect(ui->pushButton_lineclk, &QPushButton::clicked, this, &QWidget::close);
```

```
}
```

在 init 函数中对界面按钮进行 connect 信号与槽，如果提交成功，则出发 submit 函数，并且发出信号，如果是 add_station 或者 add_edge，则会触发 draw_subway_system 的槽函数，重新画画板。

```
emit done();
```

2.4 逻辑结构与物理结构

1.逻辑结构：如 2.3.1。集合结构用于记录站点所属线路等等不可重复、对顺序无要求的信息。线性结构的使用非常广泛，许多地方使用了 QVector、QList 等线性结构，如线路包含的站点、返回查询的线路结果等等。树形结构没有显式进行使用，但是 Qt 的图形对象都设置了 parent，整个图形界面实际上在逻辑上反映为一颗很大的对象树，MainWindow 是根节点。地铁网络图的表示使用了图形结构，Dijkstra 算法也是基于图形结构实现的。

2.物理结构：顺序存储、链式存储、散列存储、索引存储均有涉及。前面两项分别对应 QVector 和 QList 的使用。对于需要经常随机访问而很少插入、删除的数据列表而言，使用顺序存储结构；对于需要经常插入。删除的数据而言，使用链式存储结构。由于站点、线路的名称与其本身一一对应，故在地铁系统类中，使用名称到内容的 Hash 表来进行存储。这里用到了散列存储方法，使用了 Qt 中的 QMap 结构。关于线路中包含的所有站点，使用了索引存储结构进行存储。由于地铁系统类中已经存储了站点信息，故线路中不再重复存储，仅记录了其在地铁系统类的 Hash 表中的相应索引，减少了重复存储，提高了效率。

2.5 开发平台

如 1.5

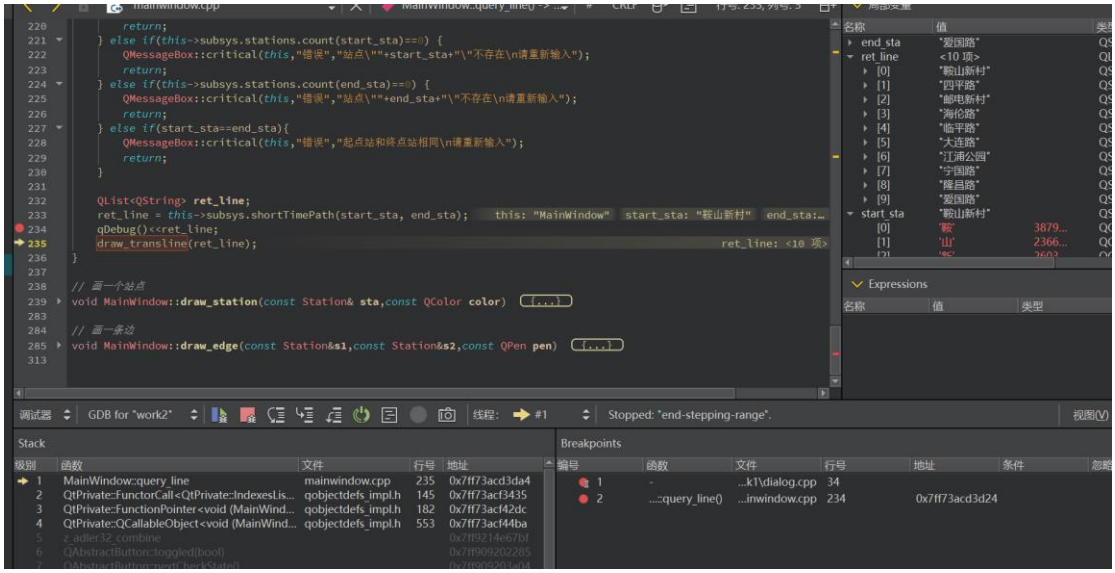
2.6 系统的运行结果分析说明

2.6.1 调试与开发过程

与 1.6.1 相似，在此简要做答

通过 qDebug 进行输出

通过 Qt 的调试模式进行调试分析

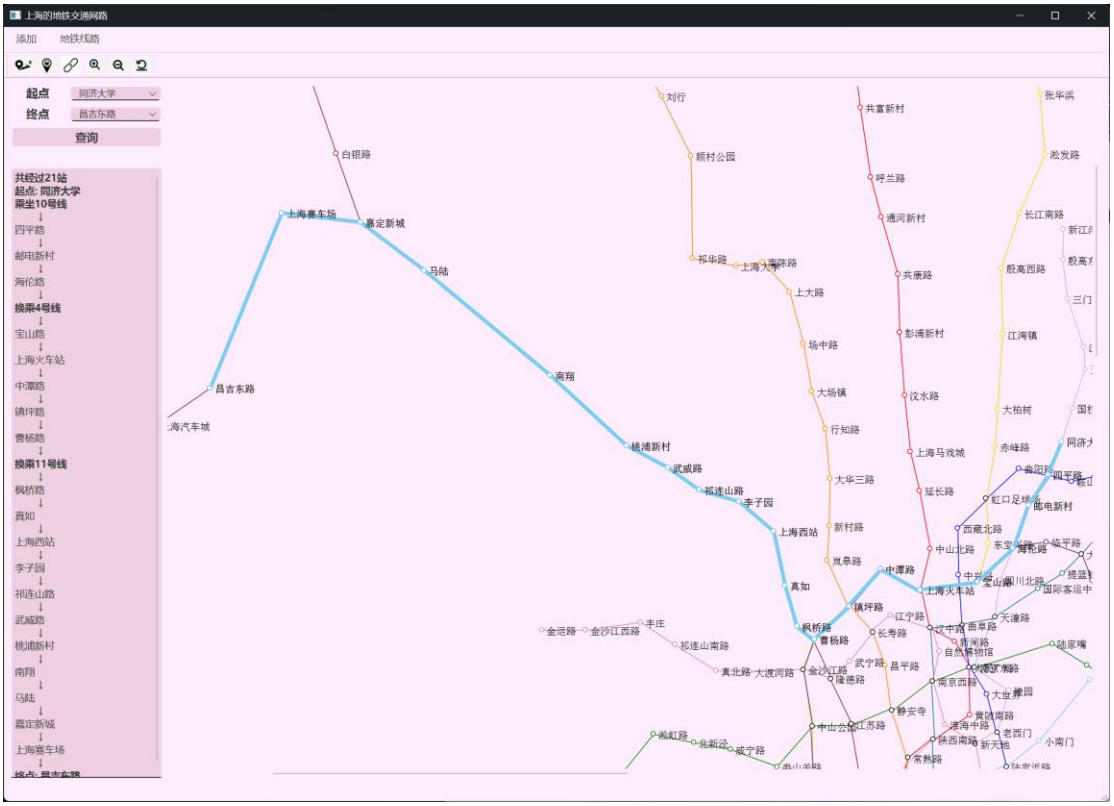


1.6.2 软件正确性、稳定性与容错能力

1. 软件正确性：

• 测试与验证：软件经过多次测试，确保在各种情况下都能按照预期执行。测试涵盖了查询和新建 line、station、edge 等各种基本操作。以下是一些测试案例：

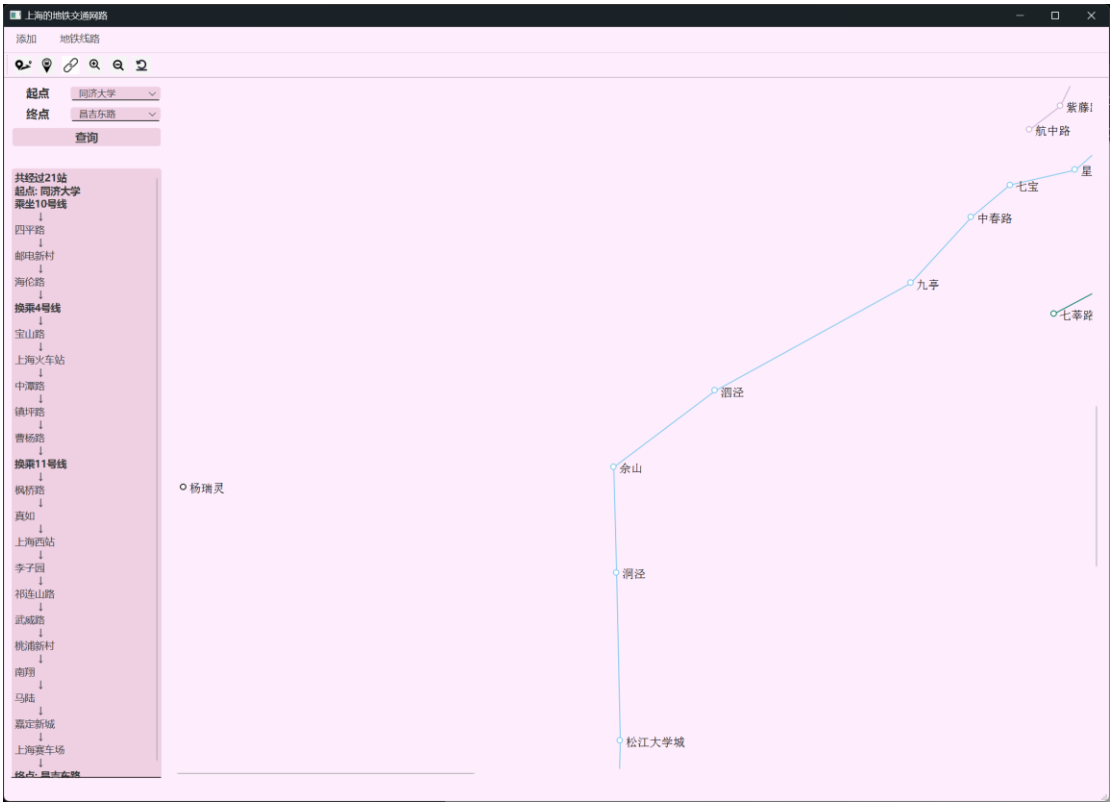
• 查询测试：同济大学->昌吉东路



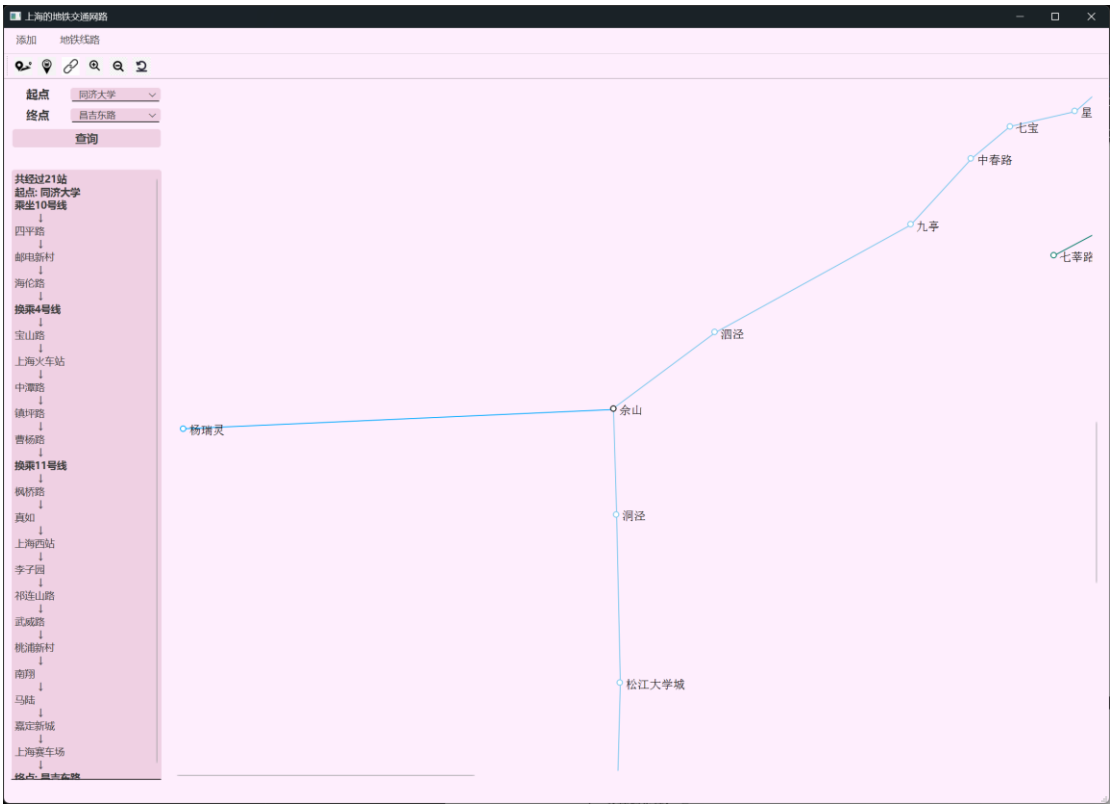
• 添加线路：


添加线路
线路名称
y
提示
线路y添加成功!
OK
确认
取消

- 添加站点：


添加
地铁线路
起点
同济大学
终点
昌吉东路
查询
共经过21站
起点: 同济大学
乘坐10号线
四平路
邮电新村
海伦路
换乘4号线
宝山路
上海火车站
中潭路
镇坪路
曹杨路
换乘11号线
枫桥路
真如
上海西站
李子园
祁连山路
武威路
桃浦新村
南翔
马陆
嘉定新城
上海赛车场
杨瑞灵
余山
洞泾
松江大学城
泗泾
九亭
中春路
七宝
航中路
七莘路

- 连接站点



2. 软件稳定性：

• 程序的稳定性表现优秀。所有的类中使用 new 运算符动态申请的内存，全部在析构函数中进行了 delete 操作。图形界面中动态申请的 Qt 类对象，全部设置了 parent。在 parent 关闭时，这些 new 运算符申请的 Qt 类对象会自动进行析构。各类消息对话框全部使用 Qt 自带的 QMessageBox 进行，避免在主窗口关闭前反复触发消息对话框可能导致的内存耗尽情况。另外，每次进行添加操作后，如果图形界面有变化，都会进行 update 操作，进行同步更新。

• 对象生命周期管理：通过在适当位置使用构造函数和析构函数，确保对象生命周期管理得当。

• 异常处理：操作中增加了异常处理机制。

3. 软件容错能力：

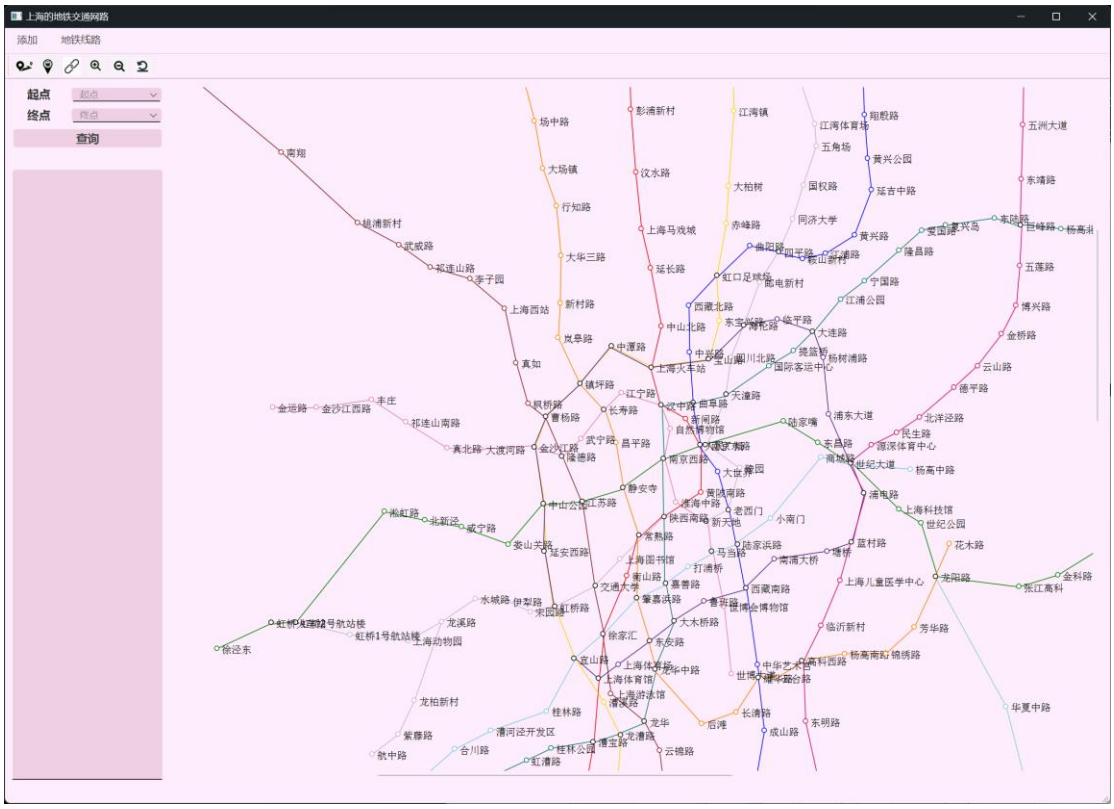
• 输入校验：查询的起点、终点不存在或者相同等会弹出对话框进行提示

```
if(start_sta=="") {
    QMessageBox::critical(this,"错误","尚未输入起点站");
    return;
} else if(end_sta==""){
    QMessageBox::critical(this,"错误","尚未输入终点站");
    return;
} else if(this->subsys.stations.count(start_sta)==0) {
    QMessageBox::critical(this,"错误","站点\""+start_sta+"\"不存在\n请重新输入");
    return;
} else if(this->subsys.stations.count(end_sta)==0) {
    QMessageBox::critical(this,"错误","站点\""+end_sta+"\"不存在\n请重新输入");
    return;
} else if(start_sta==end_sta){
    QMessageBox::critical(this,"错误","起点站和终点站相同\n请重新输入");
    return;
}
```

- 添加时输入出错会进行提示

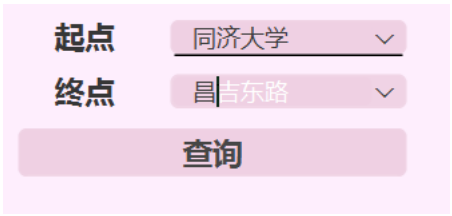
```
void SubwayControlWindow::submit_tab_line()
{
    this->name = ui->lineEdit->text();
    qDebug()<<this->name;
    qDebug()<<this->color;
    if(this->name==""){
        error_notice("未输入线路名称");
        return;
    }
    else if(this->subsys->lines.count(this->name)){
        error_notice("线路已存在,请重新输入");
        return;
    }
    else if(this->color==QColor::Invalid){
        error_notice("未选颜色！");
        return;
    }

    this->subsys->addLine(this->name,this->color);
    right_notice("线路"+this->name+"添加成功！");
    close();    //关闭窗口

    emit done();
}
```

1.6.3 运行案例

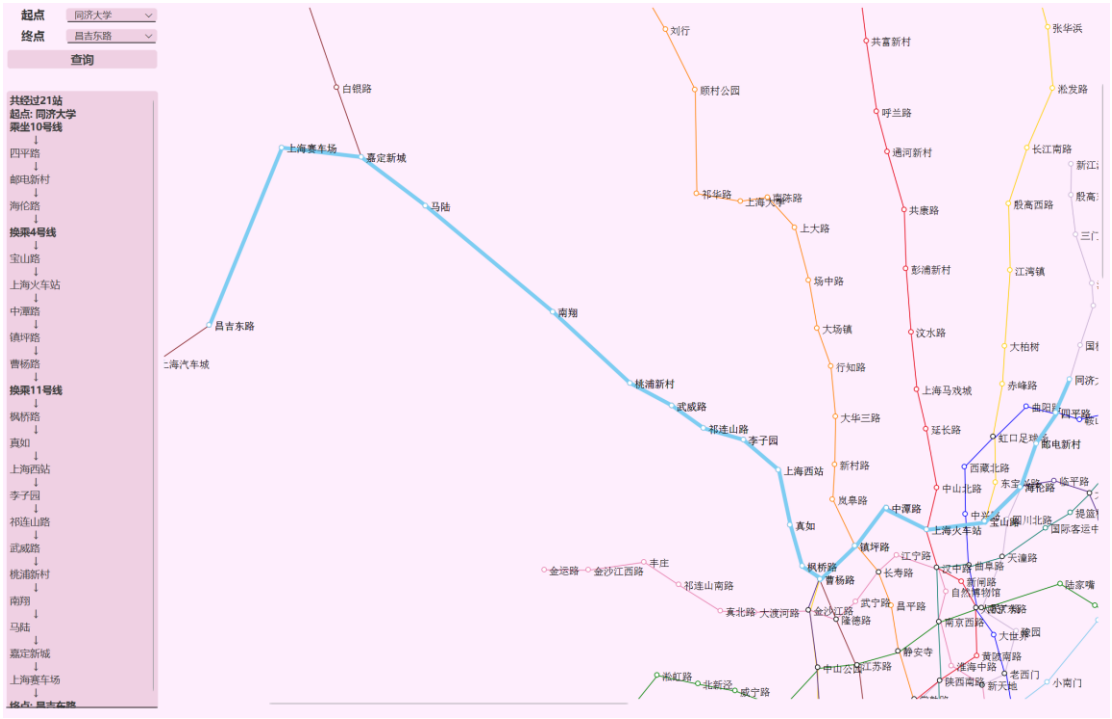


1. **查询**

提示：

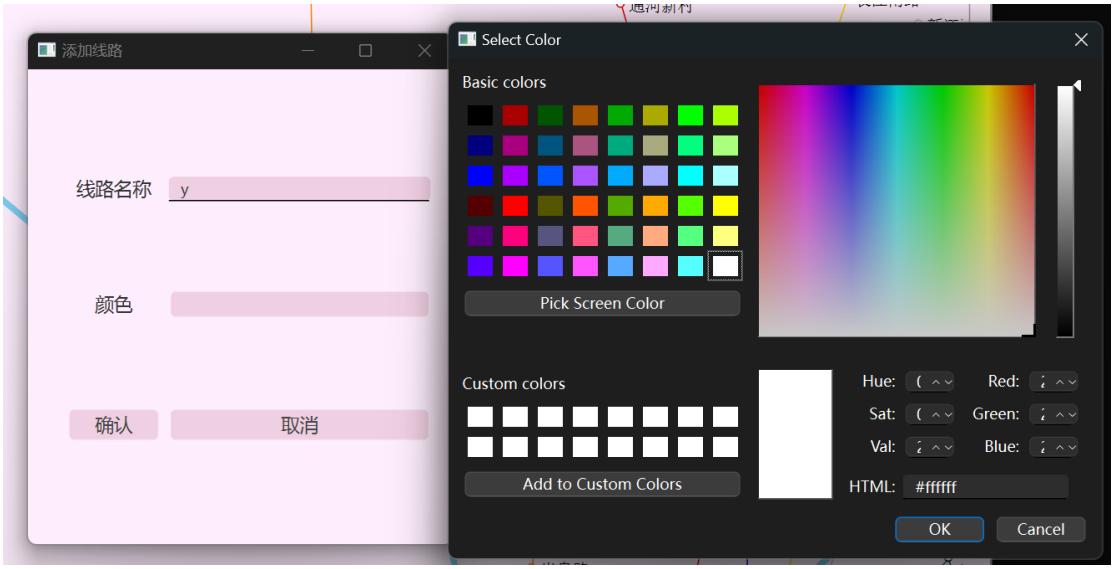


左侧显示路线，右侧显示图形化路线：



2. 添加线路、站点、连线

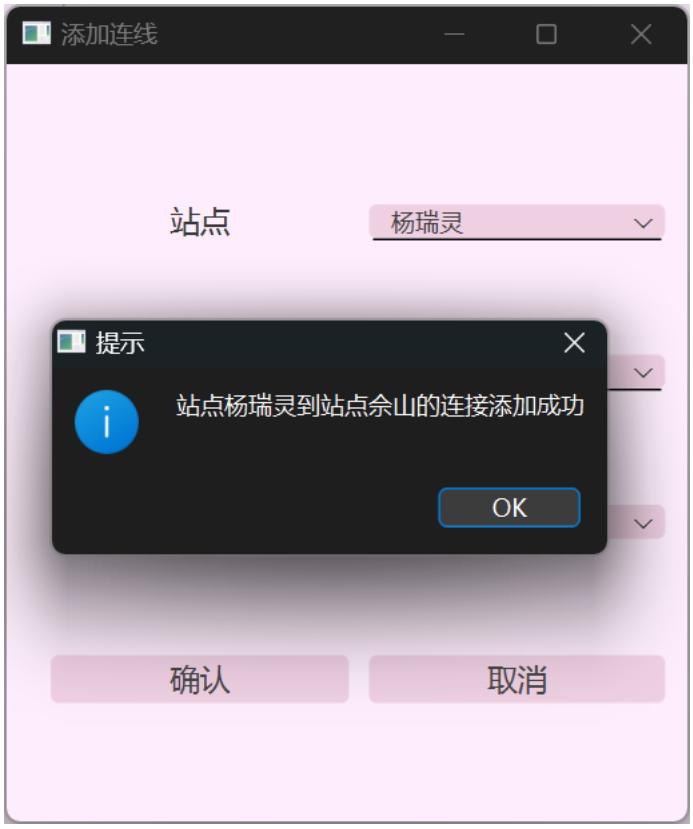
线路：

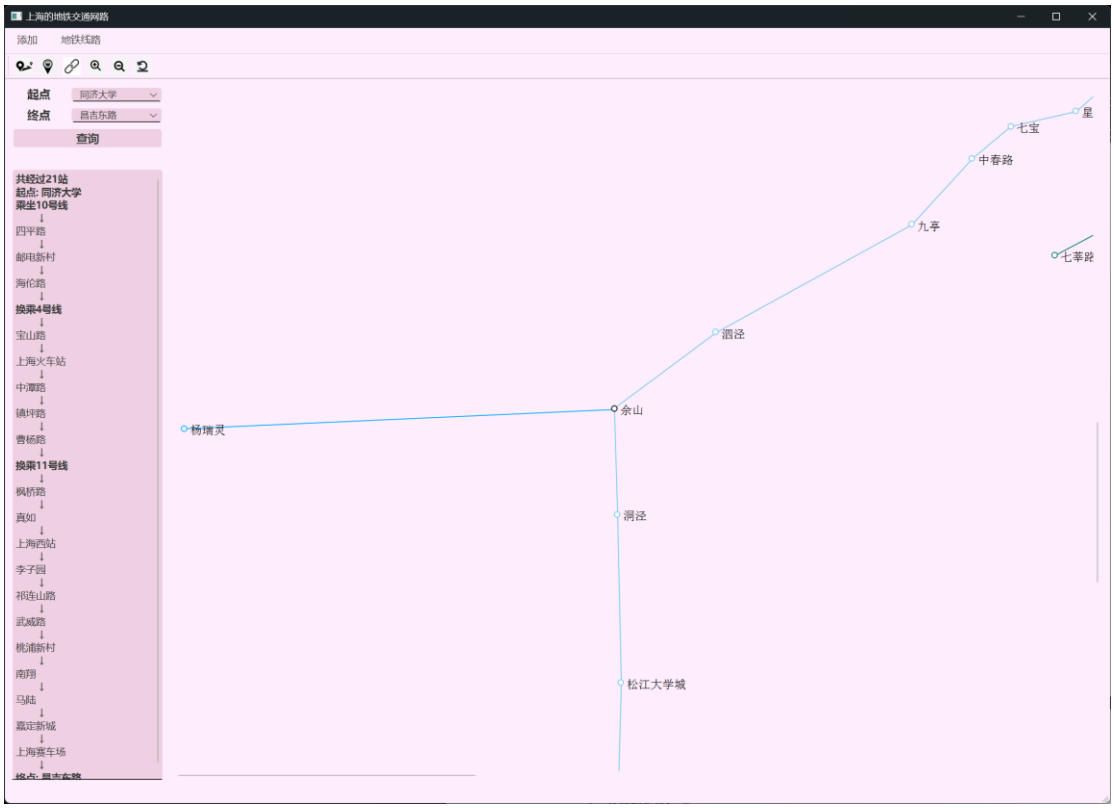


站点：


添加站点
站点名称
杨瑞灵
提示
站点杨瑞灵添加成功!
OK
经
纬度
确认
取消

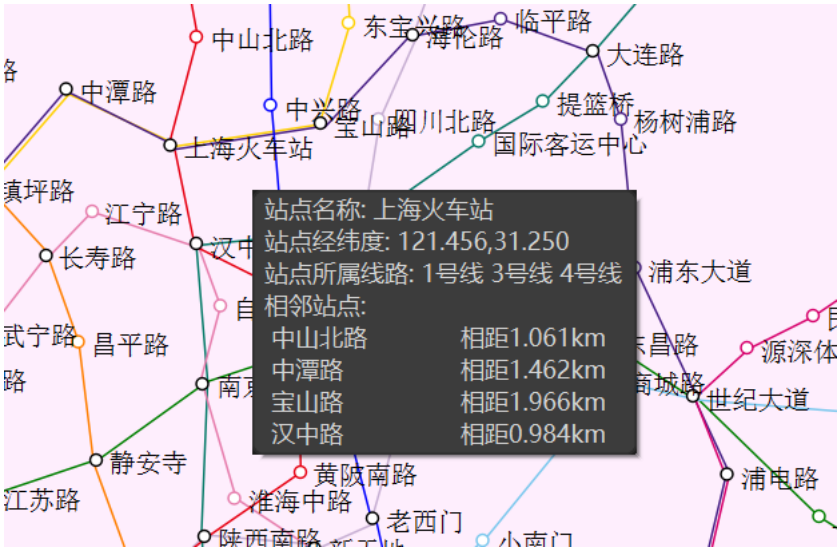
连线：


添加连线
站点
杨瑞灵
提示
站点杨瑞灵到站点佘山的连接添加成功
OK
确认
取消



3. 工具

悬停显示详细信息

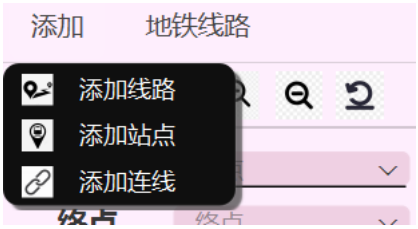


放大缩小和刷新：

2.7 操作说明

• 可以通过左上角工具栏的放大镜图标来进行地图放大、缩小、重置。放大和缩小以当前窗口的中心为中心进行。也可以通过菜单栏中的放大、缩小来进行同样的操作。

• 可以通过左上角工具栏添加图标来进行添加路线、站点和连线，也可以通过菜单栏进行添加操作。



• 选择起点和终点，点击查询可以查询路线

第三部分 实践总结

3.1. 所做的工作

• 学习掌握并且运用了 Qt5 的编程架构和相关知识： 我深入学习了 Qt5 框架的编程体

系结构，包括窗口、小部件、信号与槽机制、事件处理等。这使我能够构建具有用户友好界面的应用程序，并且有效地处理用户输入和事件。

•学习并掌握了 Qt5 的对象树与信号槽机制： Qt5 的对象树和信号槽机制是该框架的核心特性，我深入理解了它们的工作原理并成功应用于项目中。对象树的管理让我能够高效地管理内存资源，而信号槽机制则实现了模块之间的松耦合通信。

• 进一步了解了面向对象的优点并且在此次实践中灵活运用： 面向对象编程的优点包括封装、继承和多态，我在这次项目中灵活应用了这些概念，将代码组织得更清晰、可维护

性更高。

• 复习了顺序列表、循环列表和双向列表的基本操作。

• 对迪杰斯特拉算法等最短路算法进行进一步的加深理解： 在上海地铁项目中，我加深了对最短路径算法的理解，特别是迪杰斯特拉算法。我不仅理论上掌握了这些算法的原理，还成功地将它们实现应用于项目中，实现了地铁站点之间的最短路径搜索。

3.2.总结与收获

3.2.1 能力提升

在本次数据结构课程设计的实践过程中，我不仅复习了数据结构理论课中学习到的列表基本操作，图的邻接表存储方式以及 Dijkstra 单源最短路径，还从零基础地开始学习了 Qt 的图形化应用开发。

在本次课程设计的项目中，我首次接触到了 Qt 框架，学会了如何使用这一图形化编程工具。我独立自主地学习了 Qt 的信号槽机制以及对象树机制，一点一滴地在实践中提升了自己，独立完成一个项目的设计。

另外，我也学会了在使用新软件时，可以先查阅帮助文档，这一点在解决问题时非常有帮助。在面对诸如 QGraphicsView、QGraphicsItem、QPainter 等陌生的类时，我通过查阅 Qt 的官方文档，了解了它们的方法的使用方式；此外，我还通过官方文档了解了 QCombobox、QCheckbox、QPushbutton 对应的 signal 信号，对于我使用信号槽这一机制有很大的帮助。

在设计项目构架时，我学到了自顶向下的设计思想，将复杂任务分解为小块并逐步实现，这对于管理大型项目非常重要。同时，我深刻体会到了面向对象编程的优势，如良好的代码结构和封装性，这对于提高代码的可维护性和复用性至关重要。

3.2.2 课程设计的收获

首先对 Qt 编程框架的学习和掌握使我能够更自如地开发图形化应用程序。

在此之前，我只学习过通过 html 搭配 JavaScript 的方式来构建图形化界面，C++和 Qt 构建图形界面的编程方式以前对我来说是陌生的，但通过不断的实践和学习，我现在能够创建更友好、美观和用户友好的应用程序。此外，通过对 Github 开源项目托管平台的应用，我进一步了解了现代软件开发的工作流程。

其次，这次课程设计锻炼了我的独立解决问题的能力。在项目开发过程中，我遇到了各种各样的技术难题和 bug，但我坚持不懈地寻找解决方案，通过独立思考和实践，最终找到了合适的解决办法。

另外，课程设计还培养了我的项目工程思维。我学会了如何规划项目、分析需求、设计系统架构，并按计划逐步推进项目的开发。

3.2.3 个人体会

在完成本次项目的过程中，我获得了许多宝贵的体会和收获。

自学能力的重要性。尽管我之前已经掌握了一些编程知识，但在面对全新的编程框架和技术时，我需要自主学习和探索。通过查阅资料、学习文档和解决问题。

项目工程思维的价值。在整个项目周期中，我需要从项目规划、需求 分析、架构设计到编码实现，再到测试和维护，这个全面的过程让我更好地理解了项目管理 和工程化开发的概念。我学会了如何分析需求、合理规划时间、有效地分配任务。

强化了解决问题与时间管理的能力。在项目开发中，我面临了各种技术挑战和 难题。但通过不断的思考和实践，我学会了如何冷静应对问题、找到解决方案。这种解决问 题的经验不仅提高了我的编程技能，还培养了我的毅力和耐心。

第四部分 参考文献

说明在课程设计中参考的书目和论文，参考文献格式如下。

[1] Stanley B. Lippman, Josee Lajoie, Barbara E. Moo, C++ Primer 中文版, 电子工业出版社, 2013

[2] 王维波. Qt6 C++开发指南. 978-7-115-60240-4