同济大学

TONGJI UNIVERSITY

《数据结构与算法设计》

课程设计总结

学　　院　　电子与信息工程学院

专　　业　　计算机科学与技术专业

学生姓名　　吕博文

学　　号　　2151769

指导教师　　柳先辉

日　　期　　2023 年 08 月 27 日

目 录

装 订 线

装 订 线

1 算法实现设计说明

1.1 题目

题号 9：几种排序：要求随机输入一组数据，随时给出某一趟排序的变化情况

(1) 直接插入排序、折半插入排序、希尔排序

(2) 冒泡排序、快速排序

(3) 简单选择排序

1.2 软件功能

该软件功能可以分为两大类：一是排序算法的编写，编写题目中给出的六大基础排序算法；二是排序可视化的实现，这里主要通过柱状图的形式来模拟数组中的数字，经过自行对动画时间的调整，实现流畅的排序动画演示，另外，针对比较难以理解的希尔排序和快速排序，本软件程序在图形界面额外增加了辅助理解排序原理的示意图。

主要实现功能：

(1) 可以在由此菜单栏自行选择排序的种类，数据个数以及动画的操作间隔。

(2) 支持随机产生数据进行排序，同时也支持手动输入数据进行排序，并且有针对输入数据是否合法的判断。

(3) 排序开始后右侧菜单栏锁定，除"停止"按钮外其余按钮都不会响应，防止误触。

(4) 可以通过"停止"按钮随时停止整个排序。

上述功能对于六种排序均支持，具体实现借助于 QT 框架以及 C++的面向对象式编程，在之后开发平台处会详细介绍。

1.3 设计思想

1、设计准备

不同于以往的基于控制台编程的模式，本次数据结构课程设计需要我们设计出标准规范化的用户交互界面，所以本次的数据结构课程设计需要采用全新的可视化编程的方法。考虑到在众多编程语言中我对于 C++相对比较熟悉，所以经过一些资料的查询后，选择使用 QT 来实现整个程序的功能。因为 QT[3] 相对于其他可视化编程更接近 C++的编程习惯，容易上手，同时 QT 也属于较为新兴的技术框架，支持跨平台开发。所以我首先自学了 QT 的相关知识，了解了 QT 中 UI 控件的使用，最为核心的信号与槽的机制，以及视图框架的搭建等。[2]

2、设计过程

对于整个排序可视化的程序，大体上分为内部数值排序和图形界面显示两方面，在图形显示方面，我通过定义一个 *Canvas* 类来对接 Ui 类，使得 UI 指针可以直接通过调用 canvas 类中的函数进行排序，而具体的排序方法因为考虑到有六种排序，所以我选择针对每种排序实现一个类，同时在

这六个类的上层定义一个虚类 *SortObject*，具体的排序函数类继承自虚类，重写虚类的几个排序函数即可，然后再最顶层我们只需要通过调用虚类 *SortObject* 即可实现排序，这样的设计使得整体代码逻辑变得更为清晰明了。[5]

3、实现过程

设计过程中遵循的是自顶向下的设计思路，而实现时则要按照自底向上的实现方法，先实现好每个子类，进而完成整个程序的设计编写。

1.4 逻辑结构与物理结构

整个程序的逻辑结构大体与上述设计思想一致，我针对每种排序方法首先建立一个子类，这些子类继承于一个虚类函数，构成排序的部分，而负责调用绘图的则定义一个 *canvas* 类，最终这几个类在 *MainUI* 类也就是窗口类中被调用实现整个程序功能。

(1) 冒泡排序

```
#pragma once
#include "SortObject.h"
#include <QVector>
#include <QEventLoop>
#include <QVariantAnimation>
#include<QtGlobal>

//冒泡排序
class BubbleSort : public SortObject
{
    Q_OBJECT
public:
    explicit BubbleSort(QObject *parent = nullptr);

    //开始排序
    //arr排序数组，interval动画持续时间参考值
    void sort(QVector<int>data_array, int interval) override;
    //结束排序
    void stop() override;
    //绘制
    void draw(QPainter *painter, int width, int height) override;

private:
    void initArr(QVector<int>data_array,int count);

private:
    QEventLoop loop;//事件循环，方便展示动画效果
    QVariantAnimation animation;//实现属性平滑变化，方便更好的显示绘图结果

    QVector<int>arr;
    //for循环下标
```

```
    int arrI{0};
    int arrJ{0};
    //标记当前交换状态
    bool swapFlag{false};
};
```

实现

```
#include "BubbleSort.h"
#include <algorithm>
#include <cmath>
#include <QtMath>
#include <QTime>
#include <QPainter>
#include <QPaintEvent>
#include <QScopeGuard>
#include <QFontMetrics>
#include <QDebug>

BubbleSort::BubbleSort(QObject *parent)
    : SortObject(parent)
{
    //属性动画控制交换动画效果
    //animation.setDuration(2000);
    //属性从0变到1
    animation.setStartValue(0.0);
    animation.setEndValue(1.0);
    //以四次函数变化使得整体动画效果显得更自然
    animation.setEasingCurve(QEasingCurve::OutQuart);
    //循环一次
    animation.setLoopCount(1);
    //connect连接信号与槽，属性变化完以后推出循环，属性值每发生value的改变，发送update请求
    connect(&animation, &QVariantAnimation::finished, &loop, &QEventLoop::quit);
    connect(&animation, &QVariantAnimation::valueChanged, this, &SortObject::updateRequest);
}


void BubbleSort::sort(QVector<int>data_array, int interval)
{
    //确保在推出函数前进行相关的收尾工作
    auto guard = qScopeGuard([this]{
        setRunFlag(false);
        emit updateRequest();
    });
    Q_UNUSED(guard)

    stop();
    initArr(data_array,interval);
    setRunFlag(true);
```

```
42
43      int len = arr.length();
44      for (arrI = 0; arrI < len - 1; arrI++)
45      {
46          for (arrJ = 0; arrJ < len - 1 - arrI; arrJ++)
47          {
48              if (arr[arrJ] > arr[arrJ + 1]) {
49                  //需要交换位置
50                  animation.setDuration(interval * 3);//设置属性变化时间
51                  animation.start();
52                  swapFlag = true;
53                  loop.exec();//循环开始
54                  if (getRunFlag()) {
55                      qSwap(arr[arrJ], arr[arrJ + 1]);
56                      swapFlag = false;
57                  }
58              } else {
59                  animation.setDuration(interval);
60                  animation.start();
61                  loop.exec();
62              }
63              //数组内部完成交换后，需要发出更改信号,之后在canvas类中调用paint函数进而调用draw函数实现
    UI的重新绘图
64              emit updateRequest();
65              if (!getRunFlag()) {
66                  //running = false。表示按下了停止按钮，此时停止一切活动
67                  return;
68              }
69          }
70          if (!getRunFlag()) {
71              return;
72          }
73      }
74  }
75
76  void BubbleSort::stop()
77  {
78      setRunFlag(false);
79      animation.stop();
80      loop.quit();
81      emit updateRequest();
82  }
83
84  void BubbleSort::draw(QPainter *painter, int width, int height)
85  {
86      painter->setPen(QColor(200, 200, 200));
87      const int len = arr.length();
88      //边框距离
```

```
    const int left_space = 0;
    const int right_space = 0;
    QVector<int>item_height(len,0);//根据数组中数字的大小关系，提前将每个位置对应的高度算出
    const int text_space = 15; //文字和柱子间隔
    const int text_height = painter->fontMetrics().height();
    int pos_height = 0,neg_height = 0;
    for(int i=0;i<len;i++){
        if(arr[i]<0)neg_height = qMin(neg_height,arr[i]);
        if(arr[i]>0)pos_height = qMax(pos_height,arr[i]);
    }
    int tmp = pos_height - neg_height;
    for(int i=0;i<len;i++){
        item_height[i] = (double)abs(arr[i])*(height - text_height - text_space)/tmp;
    }
    int text_y = (double)abs(pos_height)*(height - text_height-text_space)/tmp + text_space/2;
    const int item_space = 10; //柱子横项间隔
    const double item_width = (width + item_space - left_space - right_space) / (double)len -
     item_space;
    double item_left = 0;
    QColor color;
    for (int i = 0; i < len; i++)
    {
        //色块位置x
        item_left = left_space + i * (item_width + item_space);
        //色块颜色
        color = QColor(200, 200, 200);
        //在执行排序操作的时候标记比较的两个元素
        if (getRunFlag()) {
            if (i == arrJ) {
                color = QColor(255, 170 , 0);
                if (swapFlag) {
                    item_left += animation.currentValue().toDouble() * (item_width + item_space);
                }
            } else if (i == arrJ + 1) {
                color = QColor(0, 170 , 255);
                if (swapFlag) {
                    item_left -= animation.currentValue().toDouble() * (item_width + item_space);
                }
            } else if (i >= len - arrI) {
                //已排序好的
                color = QColor(0, 170, 0);
            }
        }
        //画文字
        painter->drawText(item_left + item_width/2, text_y,
                          QString::number(arr.at(i)));
        //画色块柱子
        if(arr[i]>0){
```

```
            painter->fillRect(item_left, text_y - text_space/2 - text_height - item_height[i],
                              item_width, item_height[i],
                              color);
        }
        else{
            painter->fillRect(item_left, text_y + text_space/2 + text_height,
                              item_width, item_height[i],
                              color);
        }
    }
}

void BubbleSort::initArr(QVector<int>data_array,int count)
{
    arr = data_array;
    if (count < 2) {
        return;
    }
    arrI = 0;
    arrJ = 0;
    swapFlag = false;
    emit updateRequest();
}
```

(注：因为六种排序类的实现过程大致类似，所以这里只列出了冒泡排序类的具体实现，其余的参考代码文件)

(2) 排序虚基类

```
 #pragma once
#include <QObject>
#include <QPainter>

//排序对象父类，提供公共接口给canvas
//SortObject实际上是一个虚类，本身并不实现什么功能，主要起到把各个排序类汇总到一起的功能
class SortObject : public QObject
{
    Q_OBJECT
public:
    explicit SortObject(QObject *parent = nullptr);

    //定义三个虚函数，具体实现过程放在具体的Sort类中
    //开始排序
    //count元素个数，interval动画持续时间参考值
    virtual void sort(QVector<int>data_array, int interval) = 0;
    //结束排序
    virtual void stop() = 0;
    //绘制
    virtual void draw(QPainter *painter, int width, int height) = 0;
```

```

    //running排序状态
    bool getRunFlag() const;
    void setRunFlag(bool flag);

//信号函数的实现，实质上是一个函数声明，并不需要实现具体的函数，且返回值为void
signals:
    void runFlagChanged(bool running);//运行状态信号
    void updateRequest();//更新数据信号

private:
    //排序执行状态，=true则正在排序
    bool runFlag{false};
};
```

(3) 排序类型列表类

```
#pragma once
#include <QObject>
#include "SortObject.h"

//主要是为排序方式提供的一个类,可以实现主函数通过排序类型与SortObject的对接
class SortFactory : public QObject
{
    Q_OBJECT
private:
    explicit SortFactory(QObject *parent = nullptr);
public:
    static SortFactory *getInstance();

    //创建一个排序对象
    //row对应sortlist列表的排序方式
    SortObject *createSortObject(int row, QObject *parent);

    //返回排序方式列表，作为combobox选项
    QStringList getSortList() const;
};
```

(4)MainCanvas 类

```
#pragma once
#include <QWidget>
#include "SortObject.h"

//绘制排序的widget，通过切换sortobject来演示各种排序效果
class MainCanvas : public QWidget
{
    Q_OBJECT
public:
    explicit MainCanvas(QWidget *parent = nullptr);
```

```

    //不同的SortObejct对应不同排序规则
    int getSortType() const;
    void setSortObject(int type, SortObject *obj);
    //开始排序
    void sort(QVector<int>data_array, int interval);
    //结束排序
    void stop();

protected:
    //重写QWidget中的paintEvent虚函数，当有update信号发出时都会调用该函数重新改变窗口
    void paintEvent(QPaintEvent *event) override;

signals:
    //排序执行状态
    void runFlagChanged(bool running);

private:
    int sortType{-1};
    SortObject *sortObj{nullptr};
};
```

实现

```
#include "MainCanvas.h"
#include <QPaintEvent>
#include <QPainter>

MainCanvas::MainCanvas(QWidget *parent)
    : QWidget(parent)
{
    setAttribute(Qt::WA_StyledBackground, true);
}

int MainCanvas::getSortType() const
{
    return sortType;
}

void MainCanvas::setSortObject(int type, SortObject *obj)
{
    if (sortType == type) {
        return;
    }
    //删除旧的排序
    if (sortObj) {
        sortObj->deleteLater();
        sortObj = nullptr;
    }
```

```
26
27      sortType = type;
28      sortObj = obj;
29
30      if (sortObj) {
31          connect(sortObj, &SortObject::updateRequest, this, [this]{
32              update();
33          });
34          connect(sortObj, &SortObject::runFlagChanged, this, &MainCanvas::runFlagChanged);
35      }
36      update();
37  }
38
39  void MainCanvas::sort(QVector<int>data_array, int interval)
40  {
41      if (sortObj) {
42          sortObj->sort(data_array,interval);
43      }
44  }
45
46  void MainCanvas::stop()
47  {
48      if (sortObj) {
49          sortObj->stop();
50      }
51  }
52
53  void MainCanvas::paintEvent(QPaintEvent *event)
54  {
55      event->accept();
56
57      QPainter painter(this);
58      painter.fillRect(rect(),QColor(20,20,20));
59
60      if (sortObj) {
61          sortObj->draw(&painter, width(), height());
62      }
63  }
```

(5)MainUI 类

```
1  #pragma once
2  #include <QMainWindow>
3  #include<QVector>
4  #include<QMessageBox>
5  #include <QRandomGenerator>
6
7  namespace Ui {
8  class MainUI;
```

```
}

//主窗口
class MainUI : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainUI(QWidget *parent = nullptr);
    ~MainUI();

private slots:
    void on_inputOverPushButton_clicked();

    void on_randomRadioButton_clicked(bool checked);

    void on_stopPushButton_clicked();

    void on_sortPushButton_clicked();

private:
    void init();

private:
    Ui::MainUI *ui;
    QVector<int>data_array;//表示要排序的数组内容，可能随机产生，也可能由输入决定
    bool is_random_data;//表示是否是随机产生的数据
};
```

实现

```
#include "MainUI.h"
#include "ui_MainUI.h"
#include "SortFactory.h"
#include "SortObject.h"
#include <QListView>

MainUI::MainUI(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainUI)
{
    //类的构造函数，实现初始化功能，主要是编写槽函数connect连接信号
    ui->setupUi(this);
    init();
}

MainUI::~MainUI()
{
    delete ui;
}
```

```
20
21 void MainUI::init()
22 {
23     data_array.clear();
24     //排序种类初始化
25     ui->sortType->setView(new QListView(this));
26     //排序类型
27     ui->sortType->addItems(SortFactory::getInstance()->getSortList());
28     //点击开始排序
29     //开始排序按钮的connect函数，连接信号与槽
30
31     //排序状态，排序时不能修改参数
32     connect(ui->canvas, &MainCanvas::runFlagChanged,
33             this, [this](bool running){
34         ui->sortType->setEnabled(!running);
35         ui->dataNum->setEnabled(!running);
36         ui->opInterval->setEnabled(!running);
37         ui->sortPushButton->setEnabled(!running);
38         ui->inputOverPushButton->setEnabled(!running);
39         ui->randomRadioButton->setEnabled(!running);
40     });
41 }
42
43 void MainUI::on_inputOverPushButton_clicked()
44 {
45     //点击完成输入后调用的函数。
46     //输入完成按钮被按下，说明我们选择自行输入数据
47     //当输入完成按钮按下后，我们自动忽略速记数据开关的状态，也即是自行输入数据的优先级要高于随机数据
       产生的优先级
48     //读取输入框里的数据
49     QString s = ui->inputLineEdit->text();
50     QStringList t = s.split(" ");
51     data_array.clear();
52     for(auto x:t){
53         int num = x.toInt();
54         if(x == "0"){
55             data_array.push_back(0);
56         }
57         else{
58             if(num == 0){
59                 //返回值为0且本身输入的不是0，说明有非法字符
60                 QMessageBox::warning(this,"输入不合法提示","您输入的数组中含有非数字字符，请检查后重
       新输入！");
61                 ui->inputLineEdit->clear();//清空输入框
62                 data_array.clear();
63                 return;
64             }
65             else{
```

```
                data_array.push_back(num);
            }
        }
    }
}

void MainUI::on_randomRadioButton_clicked(bool checked)
{
    is_random_data = checked;
}

void MainUI::on_stopPushButton_clicked()
{
    //点击结束排序
    ui->inputLineEdit->clear();
    data_array.clear();
    ui->canvas->stop();
}

void MainUI::on_sortPushButton_clicked()
{
    //点击开始排序，首先检测合法状态
    if(data_array.size() == 0&&is_random_data == false){
        //既没有手动输入数据，也没有选择随机产生数据，那么弹出提示
        QMessageBox::warning(this,"提示","初始数据为空，请自行输入数据或是选择随机数据进行排序");
        return;
    }
    else if(data_array.size() == 0&&is_random_data == true){
        //这时，我们根据选择的数据个数来随机产生数据
        int num = ui->dataNum->value();
        for(int i=0;i<num;i++){
            data_array.push_back(QRandomGenerator::global()->bounded(-100,99));//这里我们不产生超过
    100的数据，防止可视化时视觉效果太差
        }
    }
    const int type = ui->sortType->currentIndex();//表示当前选择的排序类型
    if (type != ui->canvas->getSortType()) {
        //如果更换了排序类型，那么我们需要新建一个sortobject类来更新界面，进行新的排序
        SortObject *obj = SortFactory::getInstance()->createSortObject(type, ui->canvas);
        ui->canvas->setSortObject(type, obj);//设立新的sort类
    }
    //调用canvas类中的sort函数，进行排序
    ui->canvas->sort(data_array, ui->opInterval->value());
    ui->inputLineEdit->clear();
    data_array.clear();
}
```

(6)main.cpp

```cpp
#include <QApplication>
#include "MainUI.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MainUI window;
    window.show();
    return app.exec();
}
```

1.5 开发平台

开发环境

(1) 计算机型号 : 华硕天选 4

(2) 计算机内存 : 16GB

(3)CPU : i9-13900H 2.60 GHz

(4) 操作系统 : Windows 11

(5) 开发语言 : C++（C++11 标准以上）

(6) 开发框架 : QT

(7) 集成开发环境 : QT Verion 5.14.2

(8) 编译器 : Mingw 7.3.0 64 bit

运行环境

(1) 可在上述集成环境中运行

(2) 通过 QT 自带的 windeployqt.exe 将 release 文件夹中 VisualSort.exe 可执行文件需要的库文件在文件夹中补充完毕，在其他用户电脑上经过解压处理后保持压缩包内的文件不变，可以直接点击 VisualSort.exe 运行。

(3) 同时也通过 Enigma Virtual Box 工具将整个运行文件夹压缩为一整个 exe 文件 VisualSort-Box.exe, 如果要实现快速运行，也可以直接点击该文件运行测试。

1.6 系统的运行结果分析说明

1.6.1 调试及开发

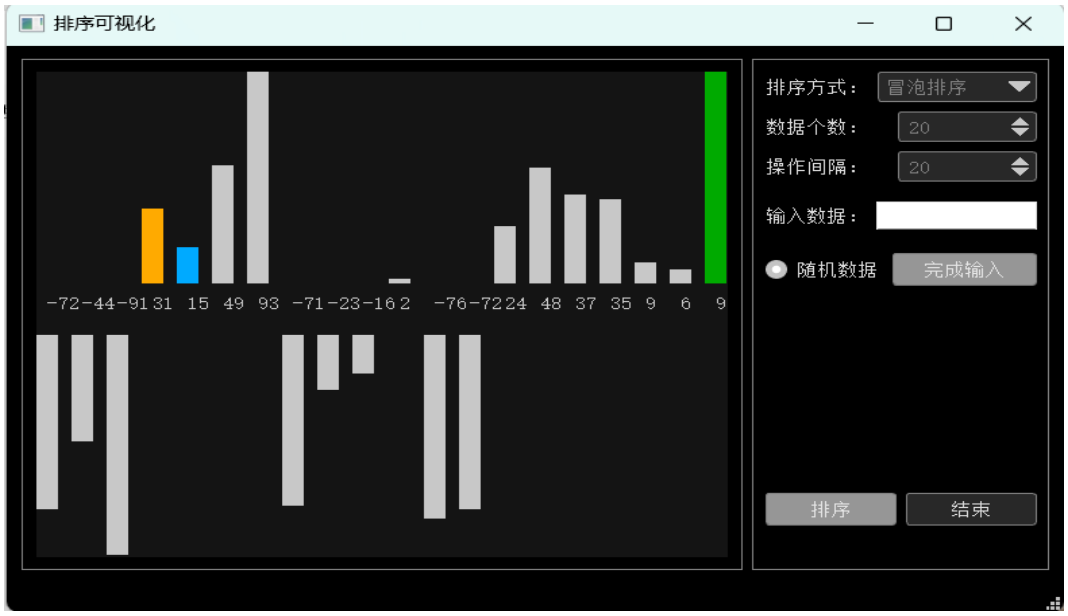
本次实验项目主要使用 QT 框架进行搭建，在 QT Creator 中进行开发调试，QT 中自带了许多原生的 C++类，使得整个开发以及调试过程变得简单，如果遇到代码运行不正确或程序异常退出的情况，我一般采用利用 QT 中自带的 *qDebug*() 函数打印错误信息，或是在代码编辑行左侧手动添加断点，在调试功能下运行。整个开发过程就是不断定义 C++类并完善的过程，最后形成整个完整的项目。[8]

1.7 成果分析

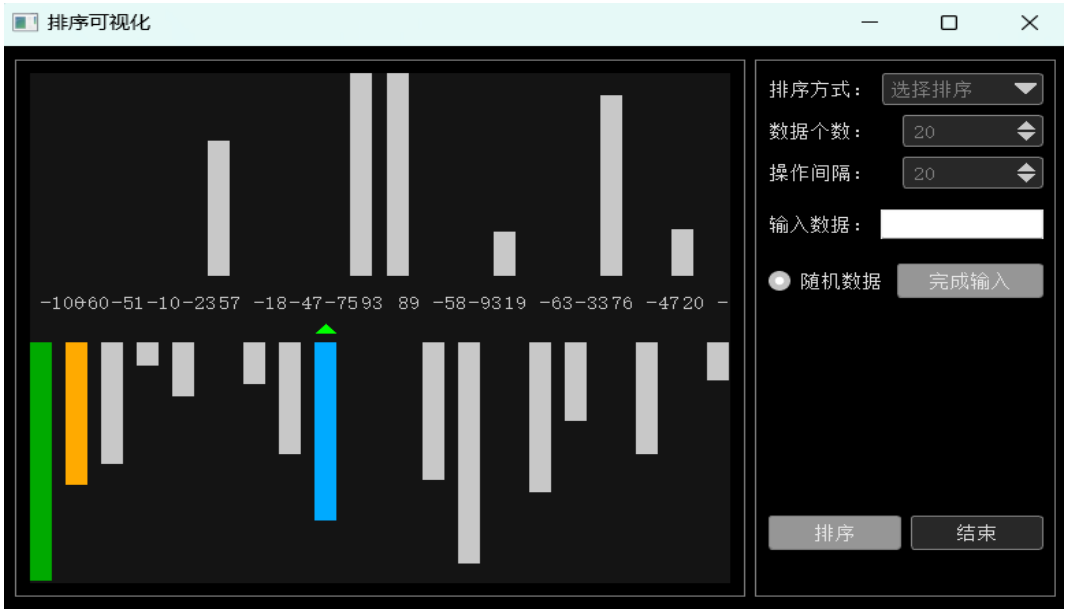
1.7.1 正确性

经过多次不同角度的手动验证，本程序表现优秀，与预期基本相符，可以保证运行的正确性，以下是基本功能的运行结果图：

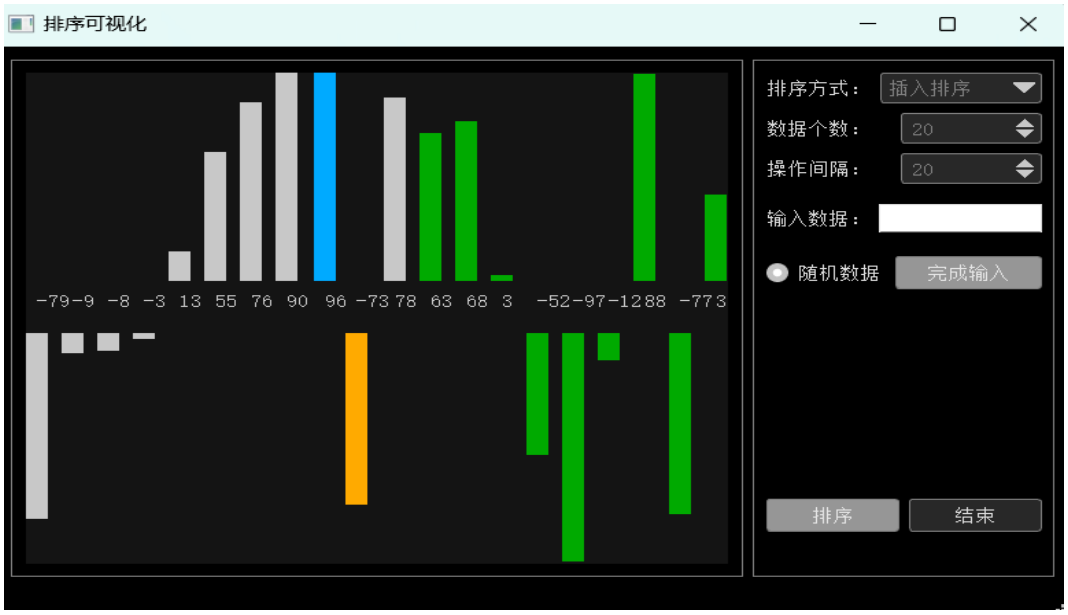
冒泡排序运行样例



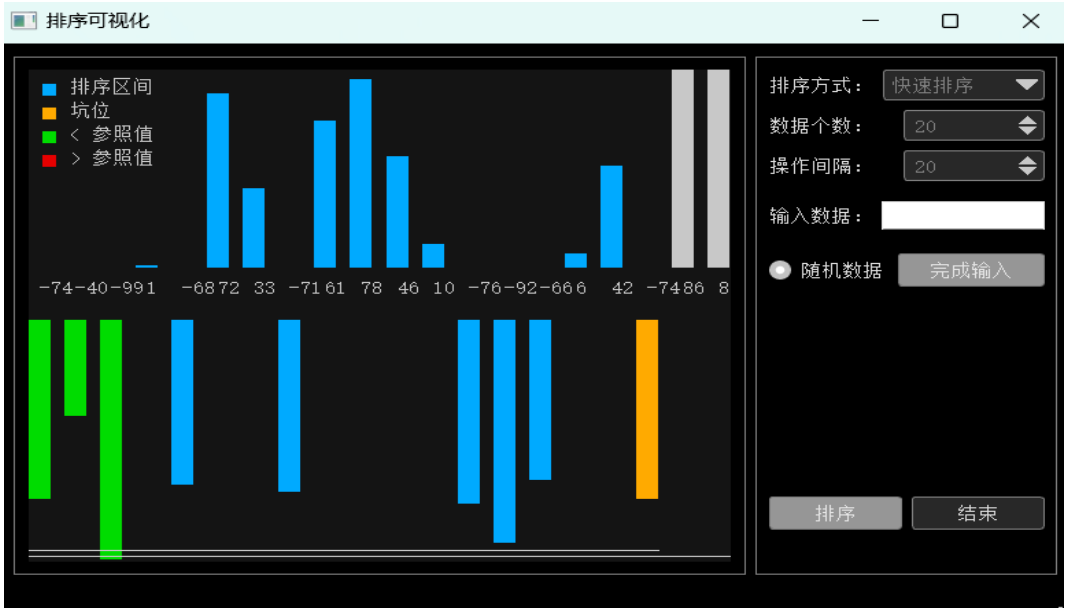
选择排序运行样例



插入排序运行样例

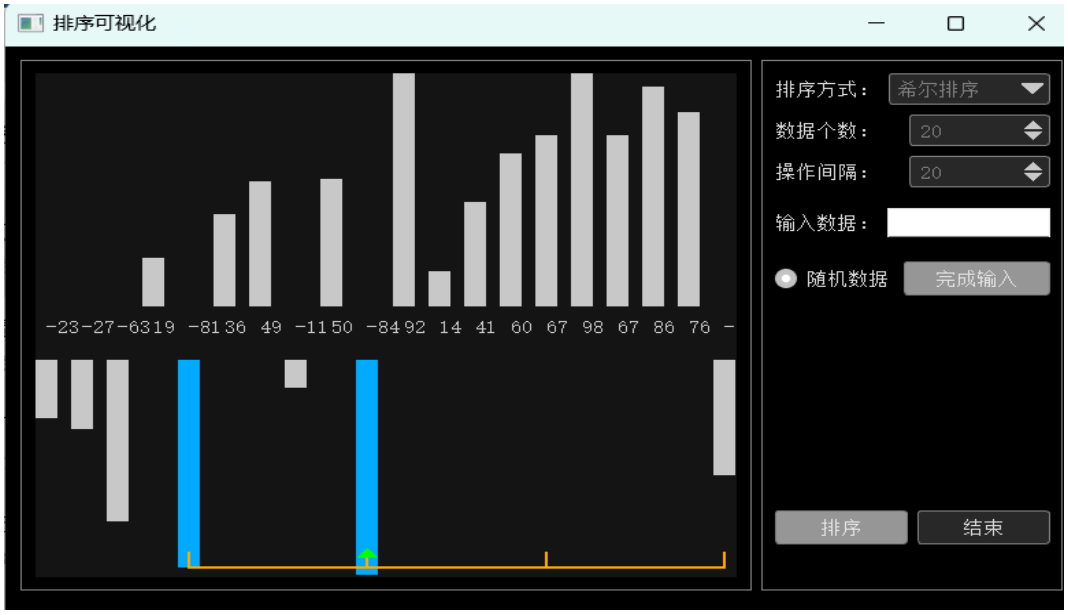


快速排序运行样例

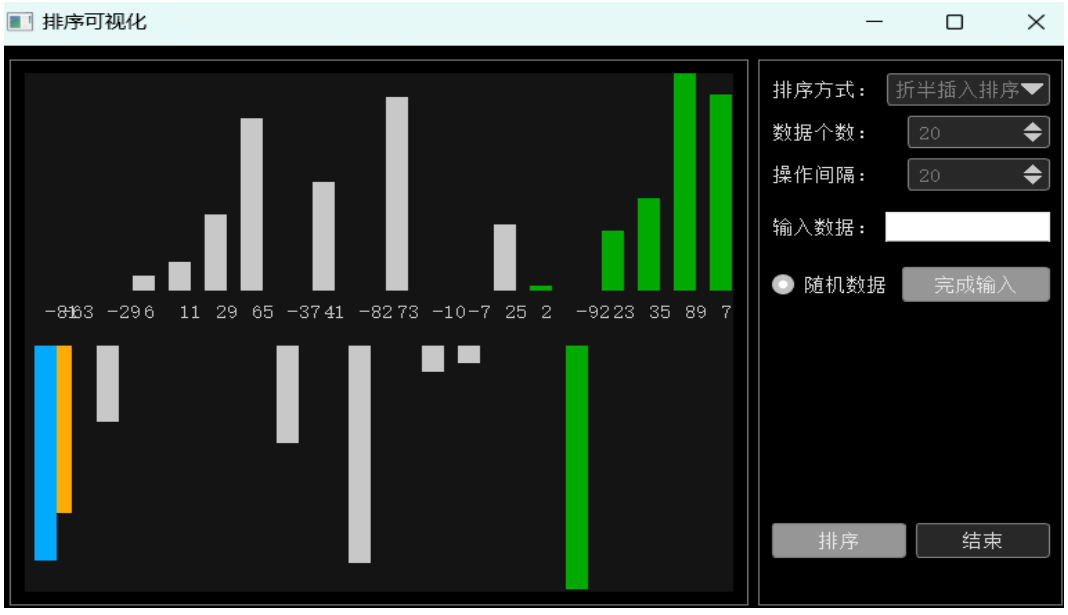


装 订 线

希尔排序运行样例



折半插入排序运行样例



装 订 线

1.7.2 稳定性

本项目整体设计基于 QT 的框架下编写，编程使用语言为 C++，库调用均为 QT 中自带的 C++语言库，程序整体架构遵循 C++面向对象的特性进行设计，着重进行类功能分块，不同的类实现不同的功能，最后集中于一个类，同时定义类时也注重数据变量的保护，数据变量尽可能定义为私有或保护类型，防止外围类直接更改类数据变量的内部值，同时程序注重变量以及函数命名的规范性，UI 界面设计的合理性，在整个程序设计和架构方面做到了稳定性的保持。

本项目在多种测试用例，多种边界条件测试下均能保持运行结果的正常输出，具体而言，针对不同的输入数据（可能有不合法的数据），不同的动画设计速度，均能实现排序可视化的正确进行，在本方面做到了稳定性的保持。

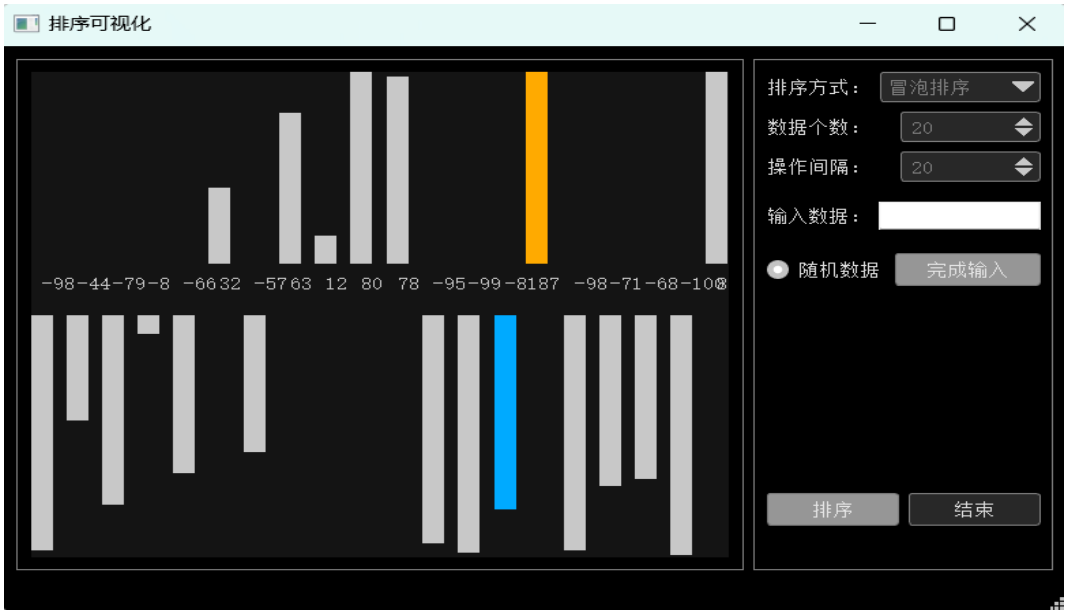
1.7.3 容错性

本项目在容错性上也做出了比较多的细节，其中六种排序功能的容错性效果相同，下面仅以冒泡排序为例进行容错性效果的检测：

(1) 对不合法的数据输入，我们会首先判断并给出提醒

如上图所示，在手动输入数据中含有不合法字符时，会给出提示，并之后清空输入框，要求重新输入。

(2) 防误触操作

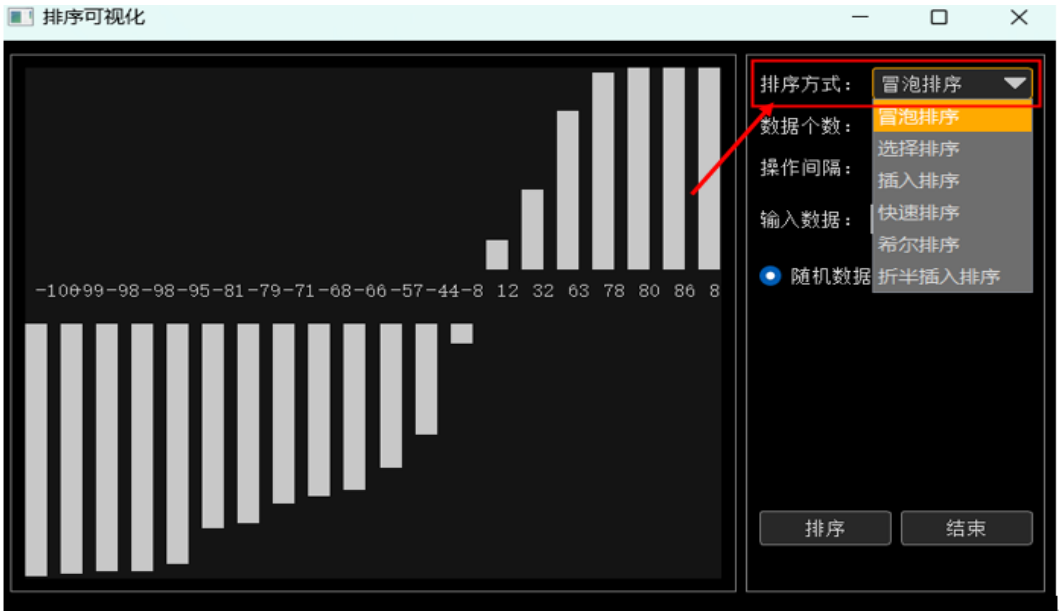


当左侧图形界面正在排序运行中时，右侧功能栏除结束按钮外，其余按钮按下不会有反应，防止误触

1.8 操作说明

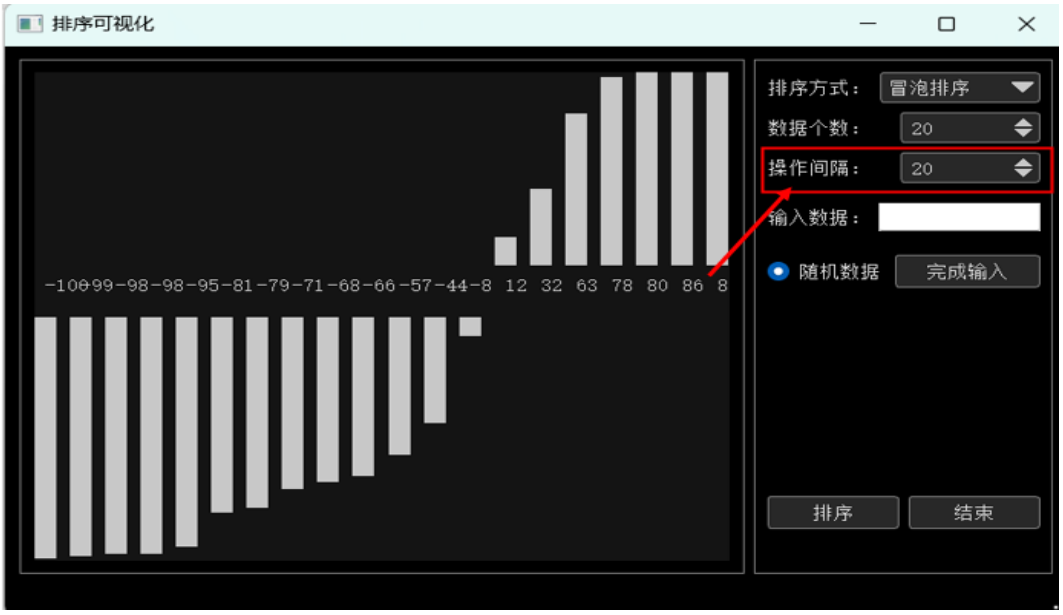
本项目实现六种基本排序的可视化功能，6 种排序的操作流程基本类似，下面以冒泡排序的整个运行流程进行介绍。

(1) 排序类型的选择



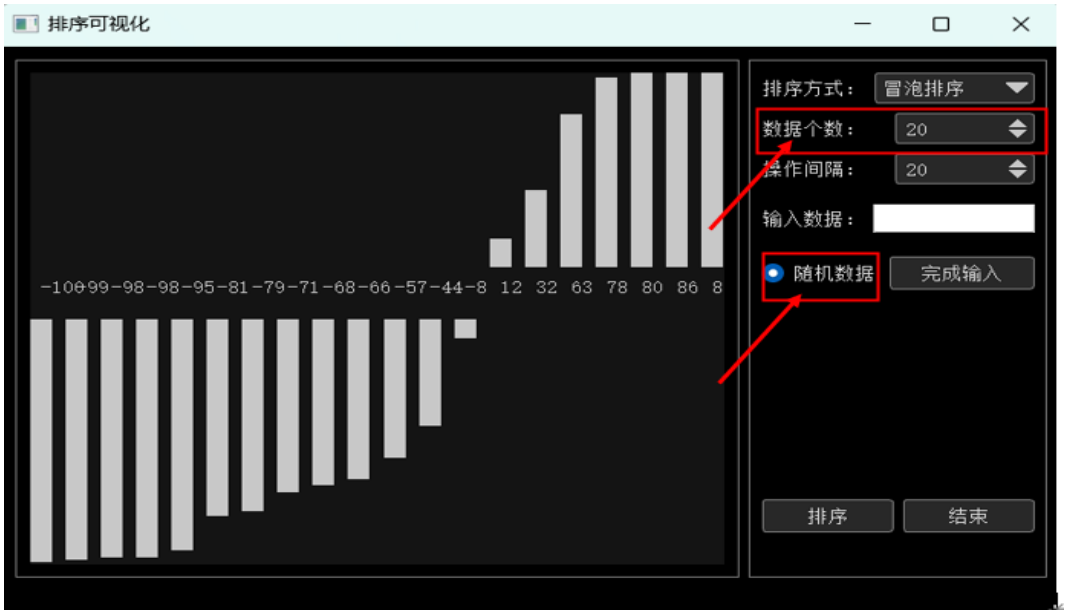
如上图所示，在界面右侧功能栏的最上方选择排序方式

(2) 操作间隔的选择



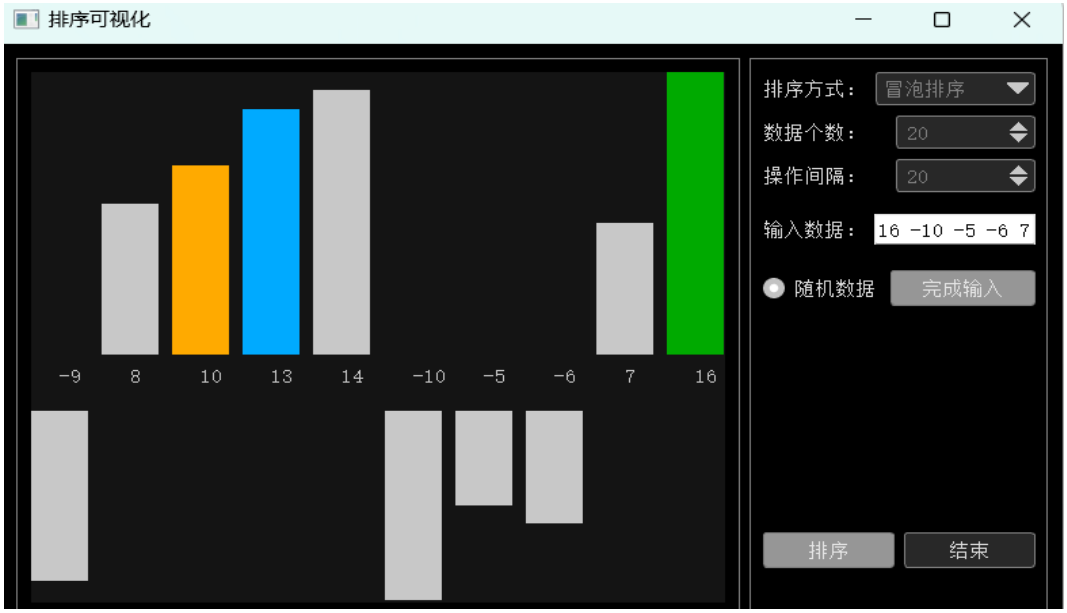
如上图所示，操作间隔代表左侧动画的速度快慢。

(3) 随机数据的产生



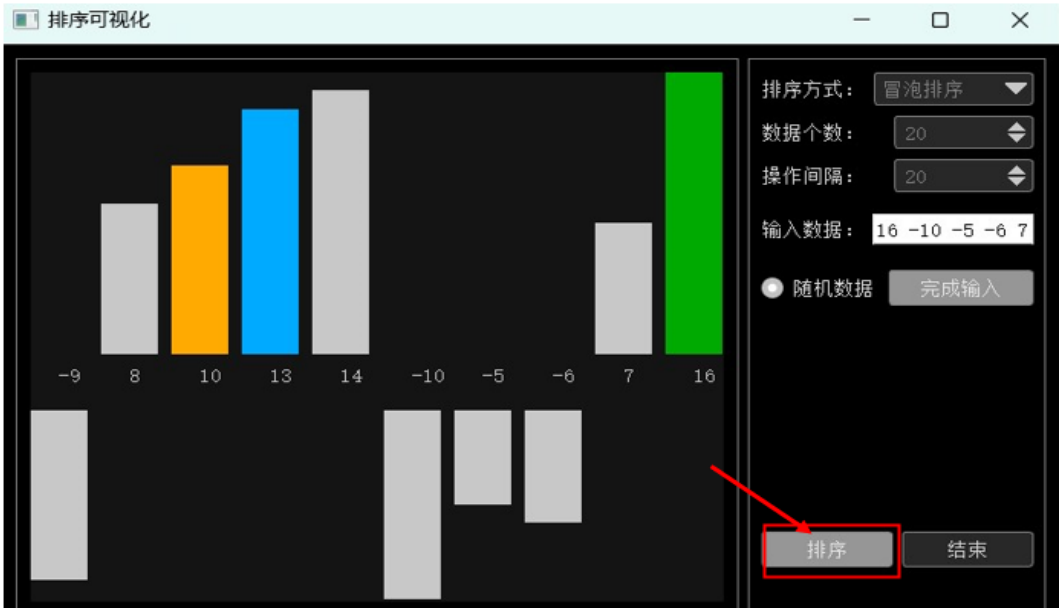
如果选择产生随机数据，需要选择随机数据的按钮，确定好数据个数。

(4) 自行输入数据 选择自行输入数据需要首先在输入框中输入所要排序的数字，点击“完成输



入”按钮，注意自行输入数据的优先级高于随机产生数据，且最后可视化排序的数据个数取决于手动输入的数据，与上方数据个数的选择无关。

(5) 排序



2 综合应用设计说明

2.1 题目

题号 2：上海的地铁交通网络已经基本成型，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

软件主要实现上海市地铁网络视图的构建，图形化展示上海市地铁网络系统，根据起点站与终点站计算地铁换乘指南，动态添加地铁线路和地铁站点以实现新站点与旧站点之间的连接。[6]

线路图查看

(1) 查看上海市地铁网络的线路图（包括每条单独的铁路线和整个地铁网络），支持鼠标拖动查看，鼠标缩放，按钮缩放等功能。

(2) 可以实时在地铁网络视图中查看各个站点和线路的具体信息。

换乘指南

(1) 支持“换乘最少”与“路程最短”两种形式的换乘指南，在下方给出具体的文字型换乘路线，并实时在左侧显示换乘的具体线路（乘车路径）

动态添加线路

(1) 支持线路新增，站点新增，连接新增的功能，并且每种新增功能都支持预览功能，可以事先查看所选择位置是否与所想一致，实现更合理化的添加功能。

2.3 设计思想

本项目还是可以主要分为前端界面的实现以及后端数据算法的维护两大方面，在前端界面的实现方面，我选取了 QT 自带的 *QGraphisView* 类进行绘图，主要实现位于 *mainwindow* 类中，这个类主要就负责图形的处理，前端信号的接收以及后端数据的呈现；对于后端数据算法，我首先针对地铁站点和地铁线路分别实现了两个类 *Station* 和 *Route* 类来管理，最后通过 *SubwayGraph* 类来集成这两个类并实现线路查找的算法。[4]

在数据结构的选取方面，首先我们可以把整个地铁网络抽象成一个无向图，站点代表图的顶点，线路代表图的边，通过图的方式去存储整个地铁线路有利于我们之后利用图论算法去求解换乘问题，同时考虑到地铁中的线路和站点都用字符串表示，为了方便，我们同时还利用了 QHash 的数据结构将站点名，线路名映射为 int 型值，方便后续的算法实现。[7]

在换乘算法方面，主要是最短路线和最少换乘路线的确定。

(1) 最短路线：我们利用图论中的 Dijstra 算法，利用图中所有点到起点的距离的大小顺序按照小根堆的形式依次弹出并利用松弛操作去更新当前点到起点的最短距离，同时每次松弛更新，我们利用一个记忆化数组 path 记录当前节点的父节点，依次进行下去我们就可以找到从起点到终点的最短路径。[1]

(2) 最少换乘路线: 首先利用 bfs 的思想从起点出发依次往外遍历相邻节点，利用一个数组记录当前线路是否被访问过，优先寻找被访问过的线路，最终我们可以求得起点到终点以及中间所有的换乘站点，在利用这些换乘站点将整个乘车路线补齐，得到最少换乘路线。

2.4 逻辑结构与物理结构

站点类定义

```
#ifndef STATION_H
#define STATION_H

#include <QString>
#include <QPointF>
#include <QSet>

class SubwayGraph;
class QTextStream;

//地铁站点类定义
class Station
{
protected:
    int id;                     //站点ID
    QString name;               //站点名称
    double longitude, latitude; //站点经纬度
    QSet<int> routesInfo;        //站点所属线路,一个站点可能属于多个线路, 这里存储的也只是线路的hash
     值

    //所有站点的边界位置
    static double minLongitude, minLatitude, maxLongitude, maxLatitude;

public:
    //构造函数
    Station();
    Station(QString nameStr, double longi, double lati, QList<int> linesList);

protected:
    //求取站点间实地直线距离
    double distance(Station other);

```

```
    //声明友元
    friend class SubwayGraph;
    friend class QTextStream;
};

#endif // STATION_H
```

线路类定义

```
#ifndef LINE_H
#define LINE_H

#include <QString>
#include <QColor>
#include <QPair>
#include <QSet>
#include <QVector>

//定义边类型
typedef QPair<int,int> Edge;

class SubwayGraph;
class QTextStream;

//线路类
class Route
{
protected:
    int id;                     //线路ID
    QString name;               //线路名称
    QColor color;               //线路颜色
    QString startStaion,dstStation;  //线路起始站点
    QSet<int> stationsSet;      //线路站点集合,这里的站点是hash化后的的int型值
    QList<Edge> edges;           //线路站点连接关系
public:
    //构造函数
    Route(){};
    Route(QString lineName, QColor lineColor):name(lineName), color(lineColor)
    {};
    //获取具有先后顺序的站点连接关系
    QList<Edge>getRouteEdges()
    {
        return edges;
    }
    //声明友元
    friend class SubwayGraph;
    friend class QTextStream;
};

```

```
#endif // LINE_H
```

地铁图结构类定义

```
#ifndef SUBWAYGRAPH_H
#define SUBWAYGRAPH_H

#include "station.h"
#include "route.h"
#include <QString>
#include <QPoint>
#include <QVector>
#include <QHash>
#include<QMessageBox>

//图的邻接点结构,方便后续使用Dijstra算法进行快速求解
class NODE{
public:
    int stationID;      //邻接点ID
    double distance;    //两点距离

    //构造函数
    NODE(){};
    NODE(int s, double dist)
    {
        stationID = s;
        distance = dist;
    };

    //">"运算重载, 用于小根堆
    friend bool operator > (const NODE&a,const NODE&b)
    {
        return a.distance > b.distance;
    }
};

//地铁视图网络类, 集成staion和route, 存储整个铁路网络的所有站点, 线路信息, 并在这个类中定义实现换乘路线求解函数
class SubwayGraph
{
protected:
    QVector<Station> stations;          //存储所有站点
    QVector<Route> routes;                //存储所有线路
    QHash<QString, int> stationsHash;   //站点名到存储位置的hash,方便staions访问
    QHash<QString, int> routesHash;      //线路名到存储位置的hash,方便routes访问
    QSet<Edge> edges;                   //所有边的集合
    QVector<QVector<NODE>> graph;       //地铁线路网络图
    QHash<Edge,QString>edgeName;        //判断该边属于哪条线路

```

```
45 public:
46     //构造函数
47     SubwayGraph();
48 
49     QString getEdgeName(Edge my_edge);
50     //获取线路名
51     QString getRouteName(int route_index);
52     //获取线路颜色
53     QColor getRouteColor(int route_index);
54     //获取线路hash值
55     int getRouteHash(QString routeName);
56     //获取线路集合hash值
57     QList<int> getRoutesHash(QList<QString> routesList);
58     //获取线路名集合
59     QList<QString> getRoutesNameList();
60     //获取线路的所有包含站点
61     QList<QString> getRouteStationsList(int route_index);
62     //获取线路的站点连接关系
63     QList<Edge>getRouteEdges(int route_index);
64 
65     //获取站点名
66     QString getStationName(int s);
67     //获取站点地理坐标
68     QPointF getStationCoord(int s);
69     //获取站点最小坐标
70     QPointF getMinCoord();
71     //获取站点最大坐标
72     QPointF getMaxCoord();
73     //获取站点所属线路信息
74     QList<int> getStationRoutesInfo(int s);
75     //获取两个站点的公共所属线路
76     QList<int> getCommonLines(int s1, int s2);
77     //获取站点hash值
78     int getStationHash(QString stationName);
79     //获取站点集合hash值
80     QList<QString> getStationsNameList();
81 
82     //添加新线路
83     void addRoute(QString lineName, QColor color);
84     //添加新站点
85     void addStation(Station s);
86     //将旧站点添加到新线路上
87     void addStationRoute(int station_index,int route_index);
88     //将旧站点到新线路上删除
89     void delStationRoute(int station_index,int route_index);
90     //添加站点连接关系
91     void addConnection(int s1, int s2, int l);
92     //删去线路
```

装 订 线

```
93      void delRoute(QString routeName);
94      //删去站点
95      void delStation(Station s);
96  //    //删去连接
97      void delConnecttion(int s1,int s2,int l);
98
99      //获取网络结构，用于前端显示
100     void getGraph(QList<int>&stationsList, QList<Edge>&edgesList);
101     //获取最少时间的线路
102     bool queryTransferMinTime(int s1, int s2,
103                               QList<int>&stationsList,
104                               QList<Edge>&edgesList);
105     //获取最少换乘的线路
106     bool queryTransferMinTransfer(int s1, int s2,
107                                   QList<int>&stationsList,
108                                   QList<Edge>&edgesList,
109                                   int&transfer_route);
110     bool my_bfs(int s1, int s2,
111                               QList<int>&stationsList,
112                               QList<Edge>&edgesList);
113     //从文件读取数据
114     bool readFileData(QString fileName);
115
116 private:
117     //清空数据
118     void clearData();
119     //插入一条边
120     bool insertEdge(int s1, int s2);
121     //更新边界经纬度
122     void updateMinMaxLongiLati();
123     //生成图结构
124     void makeGraph();
125 };
126
127 #endif // SUBWAYGRAPH_H
```

前端画图类定义

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include "graphics_view_grid.h"
5  #include "subwaygraph.h"
6  #include "managelines.h"
7
8
9  #include <QMainWindow>
10 #include <QLabel>
11 #include <QGraphicsScene>
```

```
#include <QGraphicsView>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

public slots:
    //视图放大槽函数
    void on_toolEnlarge_triggered();
    //动作视图缩小槽函数
    void on_toolShrink_triggered();
    //动作添加所有槽函数
    void on_actionAddAll_triggered();
    //动作查看所有线路图槽函数
    void on_actionLineMap_triggered();
    //动作是否显示工具栏槽函数
    void on_actiontoolBar_triggered(bool checked);
    //动作关闭程序槽函数
    void on_actionClose_triggered();


    //添加列表视图部件变化槽函数
    void tabWidgetCurrentChanged(int index);
    //添加线路功能函数
    void addRoute();
    //添加站点功能函数
    void addStation();
    //添加连接功能函数
    void addConnection();
    //线路添加预览
    void routePreview();
    //站点添加预览
    void stationPreview();
    //连接添加预览
    void connectPreview();
    //更新换乘选择信息
    void updateTranserQueryInfo();

    void add_station_preview(QPointF add_coord);

```

装 订 线

```
60
61 protected:
62     Ui::MainWindow *ui;              //主窗口UI
63     Graphics_View_Grid *myView;      //自定义视图，用于鼠标缩放
64     QGraphicsScene *scene;           //场景
65     SubwayGraph* subwayGraph;        //后端管理类
66     ManageLines* manageLines;        //添加功能前端管理类
67
68     //由线路表计算混合颜色
69     QColor getLinesColor(const QList<int>& linesList);
70     //获得线路表的名字集
71     QString getLinesName(const QList<int>& linesList);
72     //将站点的经纬度地理坐标转为视图坐标
73     QPointF transferCoord(QPointF coord);
74     //绘制网络图的边
75     void drawEdges (const QList<Edge>& edgesList);
76     //绘制网络图的站点节点
77     void drawStations (const QList<int>& stationsList);
78
79     void previewDrawEdges(const QList<Edge>& edgesList,int opt);
80
81     void previewDrawStations(const QList<int>& stationsList,int opt);
82
83 private slots:
84     void on_queryWayPushButton_clicked();
85
86     void on_routeViewPushButton_clicked();
87
88     void on_subwayAddPushButton_clicked();
89
90 private:
91     //连接信号和槽函数
92     void myConnect();
93 };
94
95 #define LINE_INFO_WIDTH 0    //预留边框用于信息展示
96 #define MARGIN 30            //视图左边距
97 #define NET_WIDTH 2000       //网络图最大宽度
98 #define NET_HEIGHT 2000      //网络图最大高度
99 #define SCENE_WIDTH (LINE_INFO_WIDTH+MARGIN*2+NET_WIDTH)    //视图宽度
100 #define SCENE_HEIGHT (MARGIN*2+NET_HEIGHT)                  //视图高度
101
102 #define EDGE_PEN_WIDTH 2     //线路边宽
103 #define NODE_HALF_WIDTH 3    //节点大小
104
105 #endif // MAINWINDOW_H
```

2.5 开发平台

开发环境

(1) 计算机型号 : 华硕天选 4

(2) 计算机内存 : 16GB

(3)CPU : i9-13900H 2.60 GHz

(4) 操作系统 : Windows 11

(5) 开发语言 : C++（C++11 标准以上）

(6) 开发框架 : QT

(7) 集成开发环境 : QT Verion 5.14.2

(8) 编译器 : Mingw 7.3.0 64 bit

运行环境

(1) 可在上述集成环境中运行

(2) 通过 QT 自带的 windeployqt.exe 将 release 文件夹中 SubwayTransferSystem.exe 可执行文件需要的库文件在文件夹中补充完毕，在其他用户电脑上经过解压处理后保持压缩包内的文件不变，可以直接点击 SubwayTransferSystem.exe 运行。

(3) 同时也通过 Enigma Virtual Box 工具将整个运行文件夹压缩为一整个 exe 文件 SubwayTransferSystemBox.exe, 如果要实现快速运行，也可以直接点击该文件运行测试。

2.6 系统的运行结果分析说明

2.6.1 调试及开发

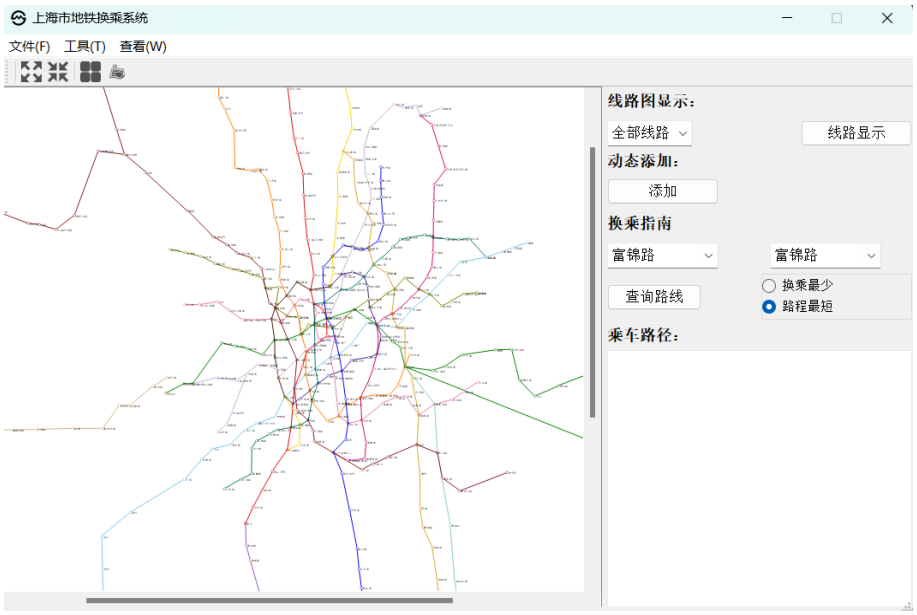
本次实验项目主要使用 QT 框架进行搭建，在 QT Creator 中进行开发调试，QT 中自带了许多原生的 C++类，使得整个开发以及调试过程变得简单，如果遇到代码运行不正确或程序异常退出的情况，我一般采用利用 QT 中自带的 *qDebug*() 函数打印错误信息，或是在代码编辑行左侧手动添加断点，在调试功能下运行。整个开发过程就是不断定义 C++类并完善的过程，最后形成整个完整的项目。[9]

2.7 成果分析

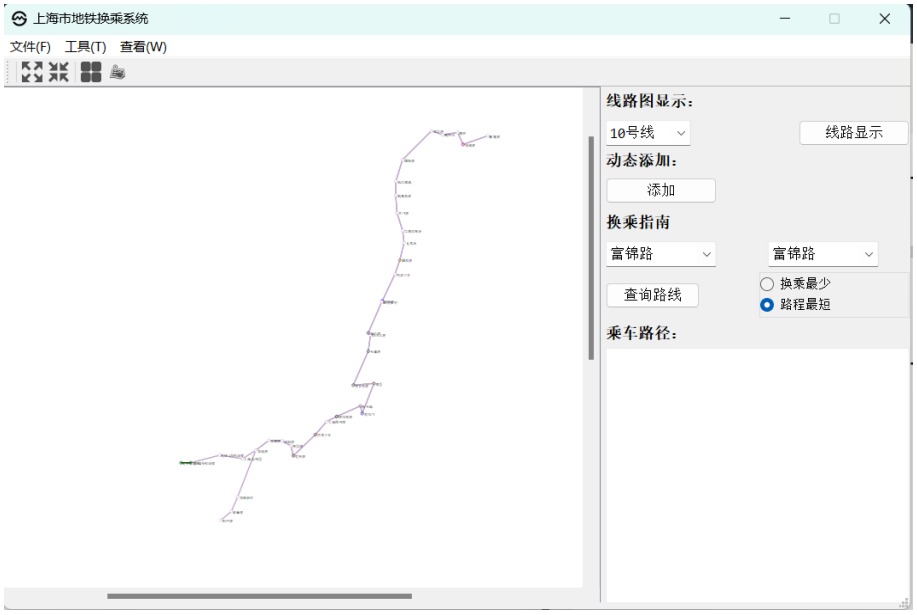
2.7.1 正确性

经过多次不同角度的手动验证，本程序表现优秀，与预期基本相符，可以保证运行的正确性，以下是基本功能的运行结果图:

(1) 初始全部地铁线路显示

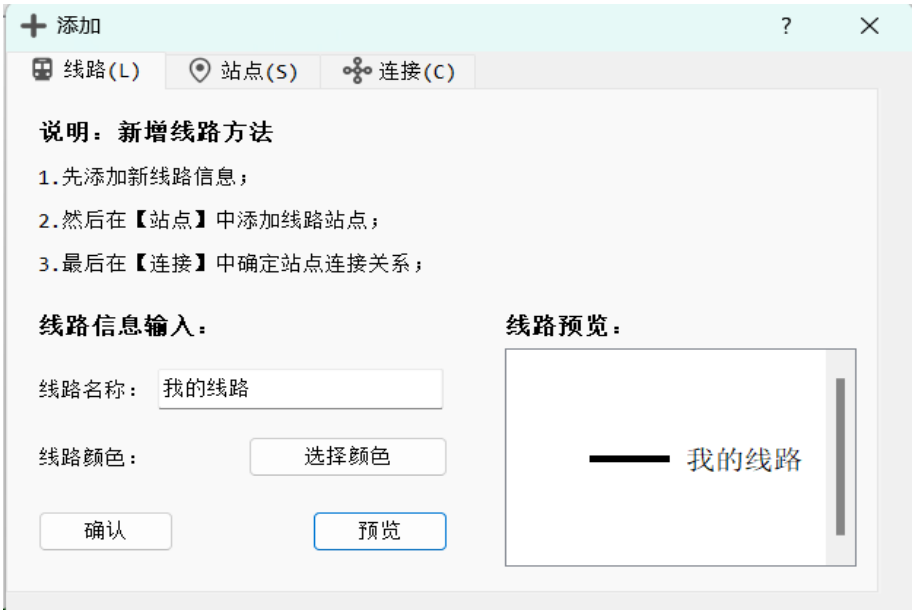


(2) 初始单独地铁线路显示



装 订 线

(3) 添加线路

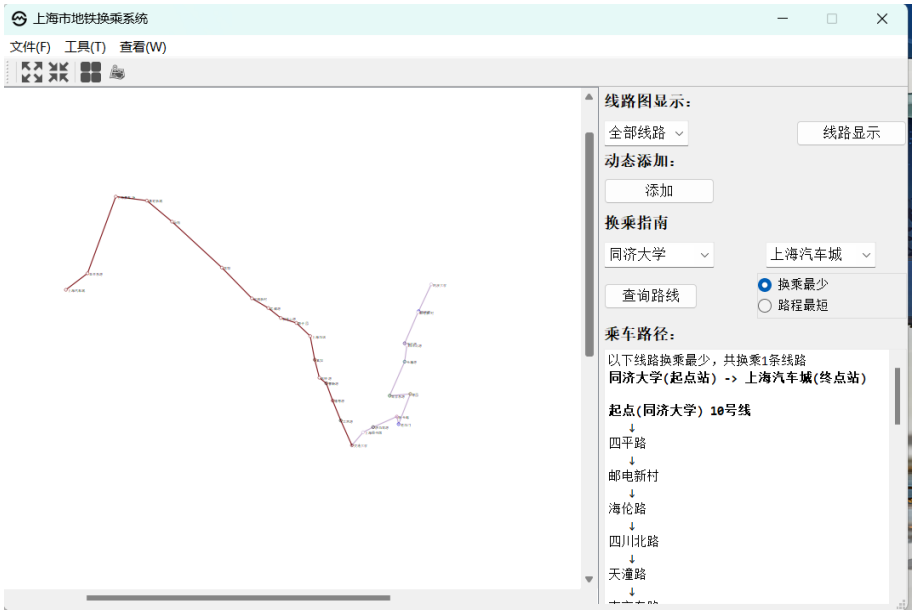

添加
线路(L)
站点(S)
连接(C)
说明：新增线路方法
1.先添加新线路信息；
2.然后在【站点】中添加线路站点；
3.最后在【连接】中确定站点连接关系；
线路信息输入：
线路预览：
线路名称：
我的线路
线路颜色：
选择颜色
确认
预览
我的线路

(4) 添加站点

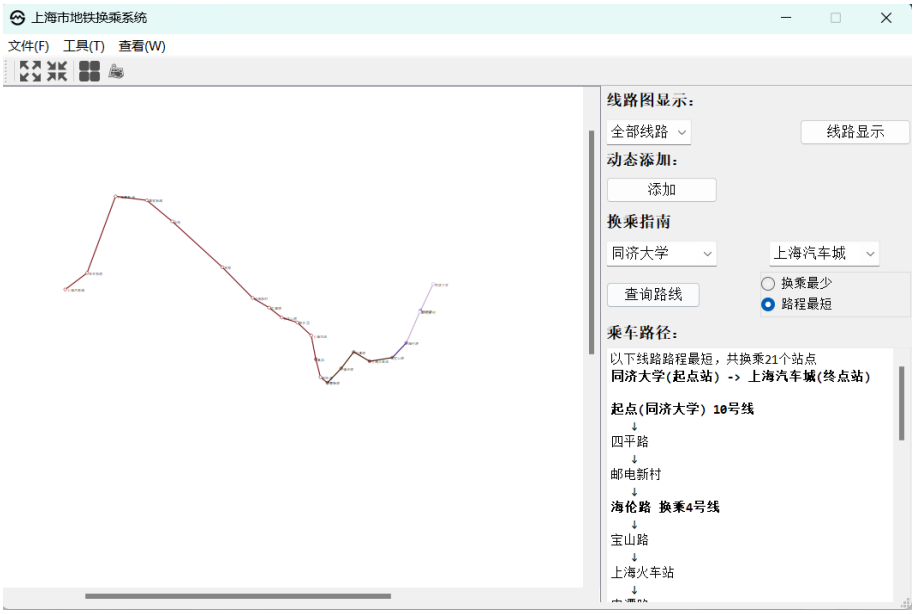

添加
线路(L)
站点(S)
连接(C)
站点信息输入：
站点位置预览：
站点名称：
我的站点
选择添加到线路：
1号线
地理坐标(P)：
东经E
121.3000000
北纬N
31.1000000
坐标有效范围：
min [121.0 , 30.9]
max [122.0 , 31.5]
确认
预览

(5) 添加连接

(6) 换乘最少路线查询



(7) 路程最短路线查询



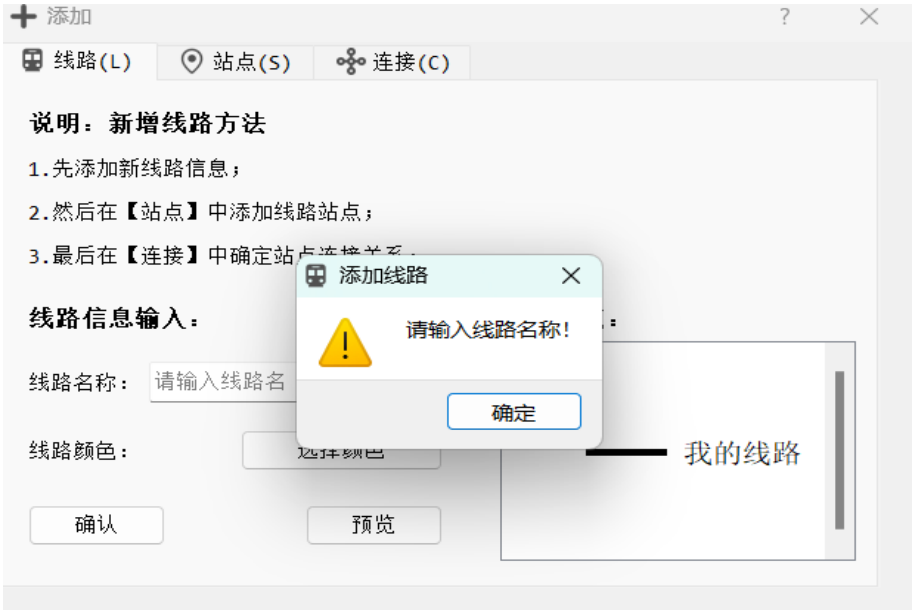
2.7.2 稳定性

本项目整体设计基于 QT 的框架下编写，编程使用语言为 C++，库调用均为 QT 中自带的 C++语言库，程序整体架构遵循 C++面向对象的特性进行设计，着重进行类功能分块，不同的类实现不同的功能，最后集中于一个类，同时定义类时也注重数据变量的保护，数据变量尽可能定义为私有或保护类型，防止外围类直接更改类数据变量的内部值，同时程序注重变量以及函数命名的规范性，UI 界面设计的合理性，在整个程序设计和架构方面做到了稳定性的保持。

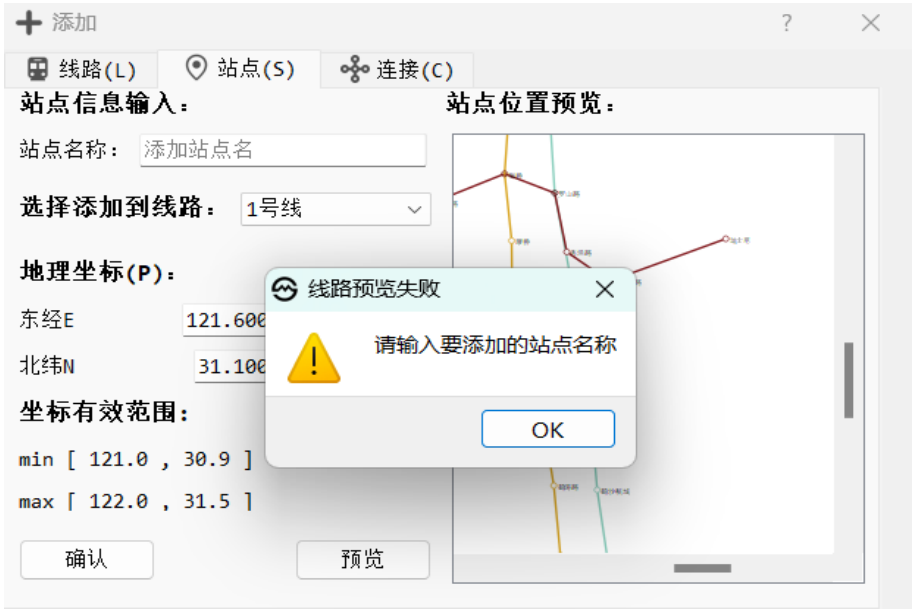
本项目在多种测试用例，多种边界条件测试下均能保持正常运行，具体而言，不管是添加新站点到新的线路还是旧站点添加到新的线路中都能够支持，对于新加入的站点，也能够支持其加入原有的线路以连接整个地铁网络系统。[10]

2.7.3 容错能力

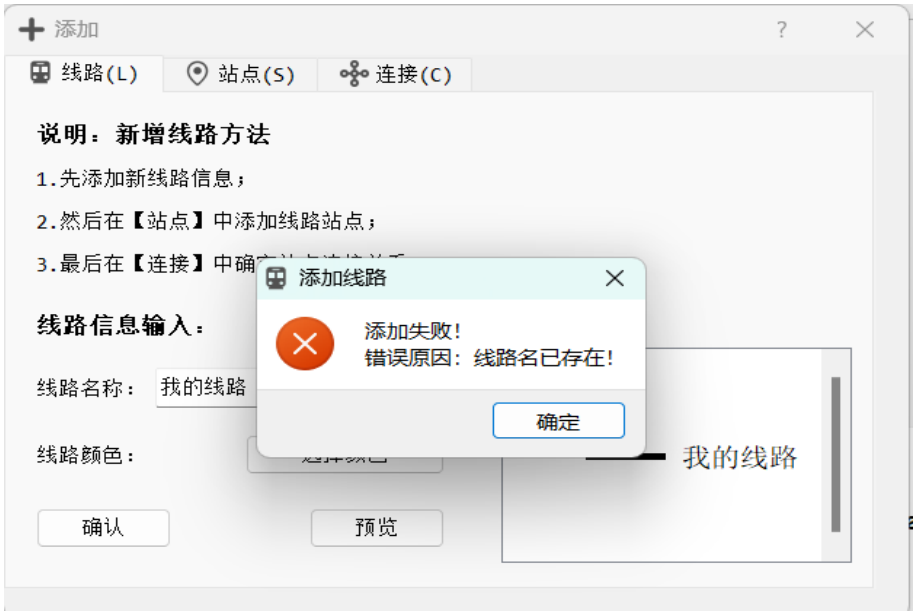
(1) 线路名称未输入提醒



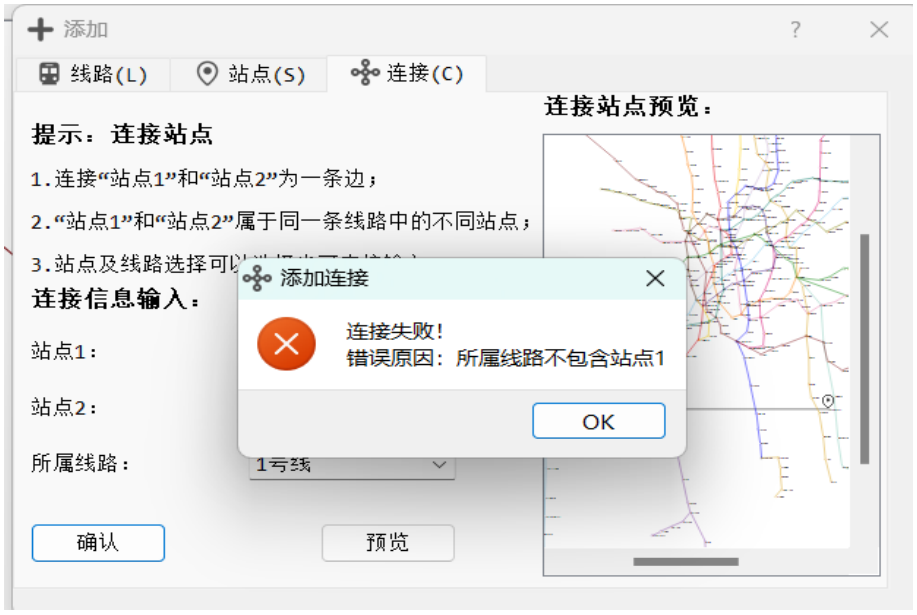
(2) 线路名已存在提醒



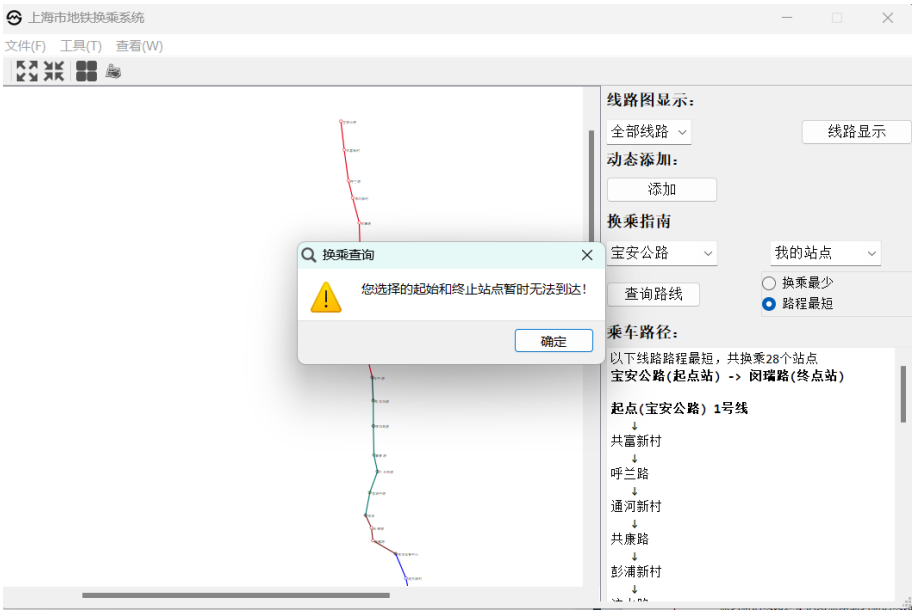
(3) 站点名未输入提醒



(4) 连接失败提醒



(5) 起始点到终止点不存在路径提醒



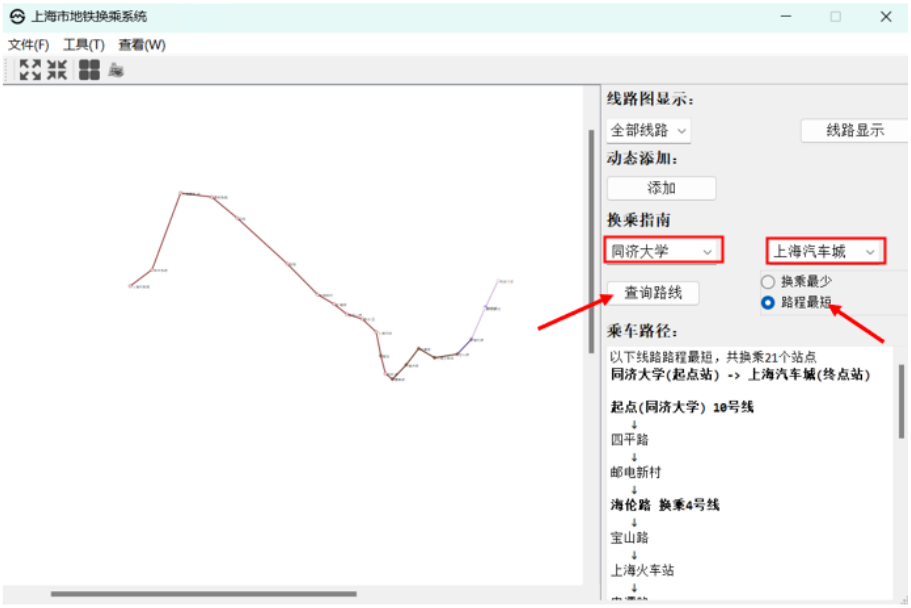
2.8 操作说明

本项目实现上海市地铁线路系统的可视化，站点线路的动态添加以及起点到终点的线路查询，具体操作细节如下：

(1) 初始地铁线路显示

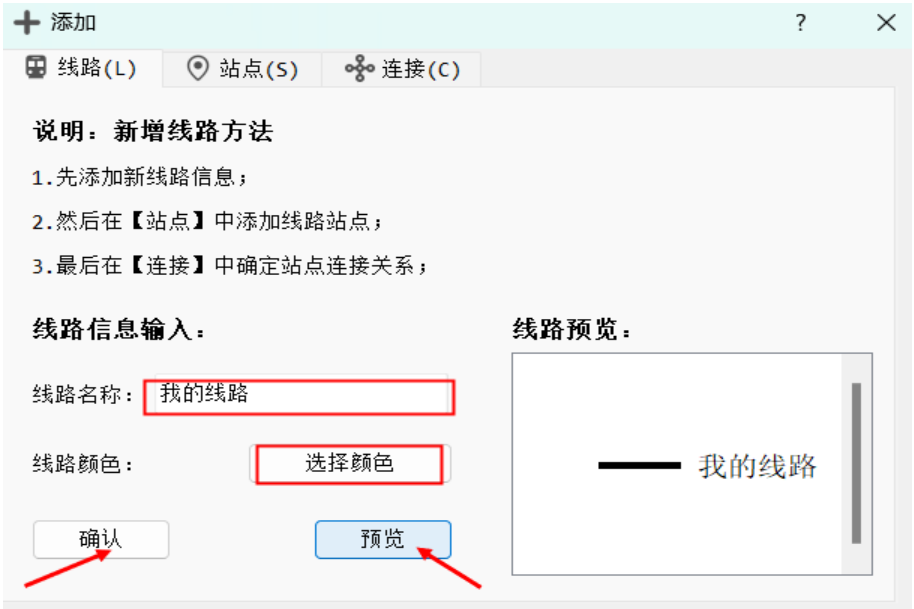
如上图所示，软件启动后，我们可以在右侧工具栏选择初始期望显示的地铁线路图，点击“线路显示”按钮后左侧即会显示相应路线

(2) 查找换乘路径



如上图所示，在右侧功能栏中有起点和终点选择框 (同时支持手动输入，含自动补全), 当点击“查询路线”按钮且输入的起点和终点合法且存在路径时，下方“乘车路径”栏就会显示具体的乘车路径，同时左侧图示栏也会显示对应的图形化换乘路径。

(3) 添加新线路



点击主界面的"添加"按钮以后进入动态添加界面，上方菜单栏选择“线路”，之后按照图示，输入新线路名称，选择颜色，点击“确定”添加，旁边还有预览功能，可以在添加之前预览效果。

(4) 添加新站点

点击主界面的"添加"按钮以后进入动态添加界面，上方菜单栏选择“站点”，之后按照图示输入新站点名称，新站点所属线路，调整输入好新站点的经纬度，同时为了便于确定添加的位置在实际地图中的位置，可以点击预览功能，通过缩放右侧图形栏，寻找图形标记来确定新站点在原有地铁线路图中的分布情况。

(5) 添加连接

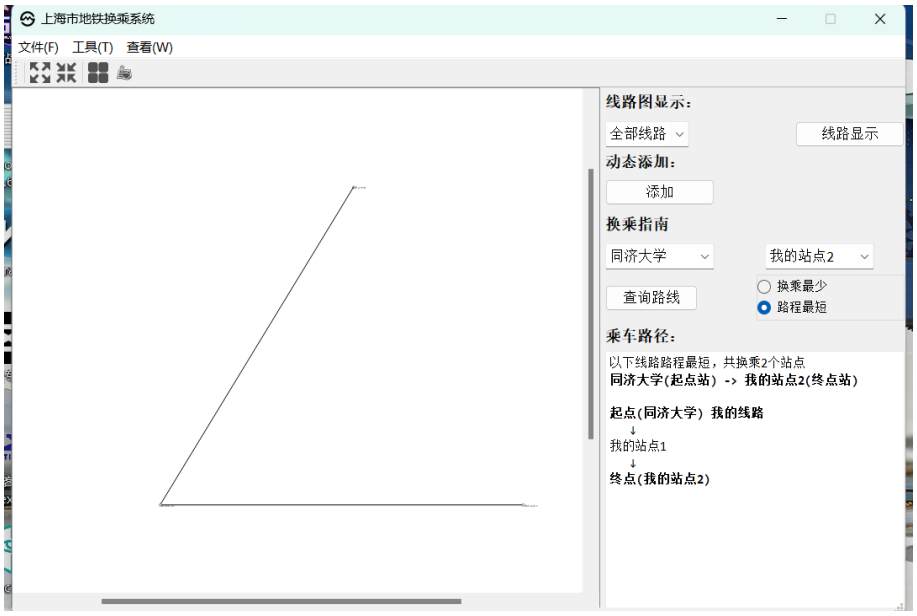
点击主界面的"添加"按钮以后进入动态添加界面，上方菜单栏选择“连接”，之后按照图示输入选择站点 1 和站点 2 以及线路，之后我们仍然可以通过预览功能预览该连接在整个地图上的分布，

之后点击确定加入连接。

(6) 新增站点与原有线路结合

首先我们将地铁原有站点“同济大学”加入新增的地铁线路“我的线路”，注意如果站点名称已存在，这时输入的经纬度将不作为输入处理，程序直接从后端数据取得该站点的真实经纬度。

之后我们将“同济大学”站和“我的站点 1”建立连接



建立连接之后我们就可以和其他站点一样进行换乘路线的计算。值得注意的是，该路线是为了方便用户直观感受软件功能而提出的示例，真实情况下添加新站点，线路连接时要考虑让新的线路与原有的线路尽可能多一些公共站点，以方便乘客换乘。

3 实验总结

3.1 所做的工作

本次课程设计一共完成了两个小型项目包括排序可视化的实现和上海市地铁换乘系统的搭建，整个课程设计采用 QT5 的编程框架，在 C++的语言基础上，结合 QT 提供的丰富的图形搭建类来实现可视化的操作。

在整个课程设计中，我学习并运用了 QT 编程的基础知识和相关拓展应用，自行上网搜集相关学习资料，解决并完成了算法实现题以及综合实现题的设计、编程、测试、封装以及最后实验报告的撰写。

3.2 总结与收获

能力提升

通过这次数据结构课程设计的实验，我的首要感觉就是自身的自学能力得到了进一步提高，在本次课程设计中，我首次被要求自行设计可视化的项目，所以在开始本项目之前，我提前在网络上搜索相关可视化的工具，网上提供了不同语言的可视化，我最终选择了以 C++为基础架构的 QT 编程架构，经过一段时间对于 QT 的基础了解，我开始着手项目实现，并在项目实现过程中不断学习更多 QT 的相关知识以辅助我完成整个项目，经过算法实现题和综合实现题，我对 QT 做到了一个

基础的入门，也为之后更加深入了解 QT 提前打好了基础。

其次，通过这个课程设计，我对于类似的大型项目设计有了更多的经验，相比与之前面对大型项目的无从下手，到现在已经可以逐步着手分析项目，拆解项目，并由小即大，最终完成整个项目的实现。

收获

通过本次课程设计，我首次接触到了标准用户交互程序，从简陋的控制台交互程序到用户化界面程序的编写，让我更一步接近了我们日常生活中使用的软件程序编写。在具体 QT 学习方面，我收获到了 QT 的大概编程框架，信号与槽机制，控件的使用，文件的读写，界面的切换等等；同时最重要的是通过本次课程设计的编写，激发了我对于编程的进一步热情以及后续学习的动力，诚然，在开发的过程中我遇到了许许多多的困难，但自行搜索资料解决这些问题并最终使得程序正常运行出结果使得我得到了很大成都的满足，也提高了我的自信心。

个人体会

通过本次课程设计，首先我感觉在完成一项大型任务时不能首先自己就给自己设置一道天堑，一定要首先坚信自己可以做到并且积极去完成，这样最终一定可以事半功倍；其次是在完成整个项目前，自己对整个项目的架构一定要清晰，否则很容易陷入不知道继续前进的困境；最后是要善于寻求帮助，不管是网络上的还是向同学询问有关某个问题，有时候别人的帮助可能会让你茅塞顿开。

装 订 线

4 参考文献

参考文献

[1] Dijkstra, Edsger W. "A note on two problems in connexion with graphs." Numerische mathematik 1.1 (1959): 269-271.

[2] Blanchette, Jasmin, and Mark Summerfield. "C++ GUI Programming with Qt 4." 2nd ed. Prentice Hall, 2008.

[3] Qt Documentation. https://doc.qt.io/

[4] Krug, Steve. "Don't Make Me Think: A Common Sense Approach to Web Usability." 2nd ed. New Riders, 2005.

[5] Jasinski, Marek. "Advanced Qt Programming: Creating Great Software with C++ and Qt 4." Prentice Hall, 2010.

[6] Chen, Wei, et al. "Design and Implementation of a Mobile App for Subway Navigation." International Journal of Human-Computer Interaction 36.4 (2020): 385-394.

[7] Blanchette, Jasmin, and Mark Summerfield. "C++ GUI Programming with Qt 5." 2nd ed. Prentice Hall, 2016.

[8] Rao, Jayaram, et al. "Introduction to Qt Quick." Qt Developer Network, 2021.

[9] Holmes, Richard. "Beginning Qt Programming." Apress, 2020.

[10] Study of QT https://www.devbean.net/2012/08/qt-study-road-2-qt-intro/

装 订 线