

《数据结构》课程设计总结

学号： 2252552

姓名： 胡译文

专业： 信息安全

2024 年 7 月

目 录

第一部分 算法实现设计说明.....	1
1.1 题目.....	3
1.2 软件功能.....	3
1.3 设计思想.....	3
1.4 逻辑结构与物理结构.....	4
1.5 开发平台.....	4
1.6 系统的运行结果分析说明.....	5
1.7 操作说明.....	8
第二部分 综合应用设计说明.....	9
2.1 题目.....	9
2.2 软件功能.....	9
2.3 设计思想.....	9
2.4 逻辑结构与物理结构.....	10
2.5 开发平台.....	11
2.6 系统的运行结果分析说明.....	12
2.7 操作说明.....	15
第三部分 实践总结.....	19
3.1. 所做的工作.....	19
3.2. 总结与收获.....	19
第四部分 参考文献.....	19

第一部分 算法实现设计说明

1.1 题目

2. 二叉树，完成：

- (1) 建立一棵二叉树, 并对它进行先序、中序、后序遍历,
- (2) 统计树中的叶子结点个数;
- (3) 分别对它进行先序、中序、后序线索化;
- (4) 实现先序、中序线索树的遍历;
- (5) 显示该树和线索化后的树（此要求可视情况选择是否完成）。

1.2 软件功能

功能分为三个板块，分别是：

- (1) 建立、计算叶子数和绘制树的基本操作。
- (2) 通过递归实现的先序、中序、后序遍历
- (3) 先序、中序、后序线索化并实现先序、中序线索树的遍历

1.3 设计思想

1. 建立二叉树

首先，我们需要定义二叉树的节点结构 BiTNode 和二叉树类 BiTree。节点结构包括数据域和左右孩子的指针，以及两个标记位 lTag 和 rTag，用于中序和后序线索化时标记左孩子和右孩子。

然后设计创建界面，按照提示输入创建字符串，注意：要求字符串长度小于 15



2. 遍历二叉树

先序遍历：首先访问根节点，然后递归遍历左子树，最后递归遍历右子树。

中序遍历：先递归遍历左子树，访问根节点，然后递归遍历右子树。

后序遍历：先递归遍历左子树，然后递归遍历右子树，最后访问根节点。

3. 统计叶子节点个数

通过递归函数 Leaves，如果节点为空，返回 0；如果节点是叶子节点（没有左右孩子），返回 1；否则，返回左子树和右子树的叶子节点数之和。

4. 线索化二叉树

先序线索化：使用 preOrderThreading 函数，从根节点开始，按照先序遍历的顺序，将空的左孩子替换为前驱节点，将空的右孩子替换为后继节点。

中序线索化：使用 inOrderThreading 函数，从根节点开始，按照中序遍历的顺序进行线索化。

后序线索化：使用 postOrderThreading 函数，从根节点开始，按照后序遍历的顺序进行线索化。

5. 线索树的遍历

先序线索树遍历：从根节点开始，按照先序遍历的顺序，使用 preOrderThreading 函数生成的线索，访问节点。

中序线索树遍历：从根节点开始，使用 inOrderThreading 函数生成的线索，按照中序遍历的顺序访问节点。

6. 清理线索

使用 cleanThread 函数，将所有线索标记位重置，并恢复二叉树的原始结构，以便下次线索化。

7. 绘制树

使用 paint 类函数，将创建后的树绘制在新窗口中。（不能绘制线索化后的树）

1.4 逻辑结构与物理结构

1.4.1 二叉树结点

```
typedef QChar ElemType;

typedef class BiTNode {
public:
    ElemType data;
    BiTNode *lchild, *rchild;
    bool lTag, rTag;
    BiTNode();
    BiTNode(QChar a);
    ~BiTNode();
}*BiTNodeP;
```

1.4.2 二叉树

```
class BiTree { //创建二叉树
public:
    BiTNode *root; //根节点
    int leafNum;
    bool ifThread=FALSE;
    QString preString, inString, postString;
    BiTree():root(nullptr){}
    ~BiTree();
    //传递节点指针的指针!!! const char*& str
    //用#表示空结点
    Status isEmpty();
    BiTNode * createBiTree(QString str);
    Status preOrder(BiTNode *r);
    Status inOrder(BiTNode *r);
    Status postOrder(BiTNode *r);
    BiTNode* NextNode2(BiTNode* p);
    BiTNode* NextNode(BiTNode* p);
    BiTNode* FirstNode(BiTNode* p);
    Status preOrderThreading(BiTNode *r);
    Status inOrderThreading(BiTNode *r);
    Status postOrderThreading(BiTNode *r);
    Status cleanThread(BiTNode *r);
    int Leaves(BiTNode *r);
    int getHeight(BiTNode *r);
    Status destroy();
    friend class Paint;
};
```

1.5 开发平台

1.5.1 开发平台

开发语言: C++ (C++11 标准以上)
开发框架: QT
集成开发环境: Qt Version 6.5.3
编译器: MinGW 11.2.0 64-bit for c++

1.5.2 运行环境

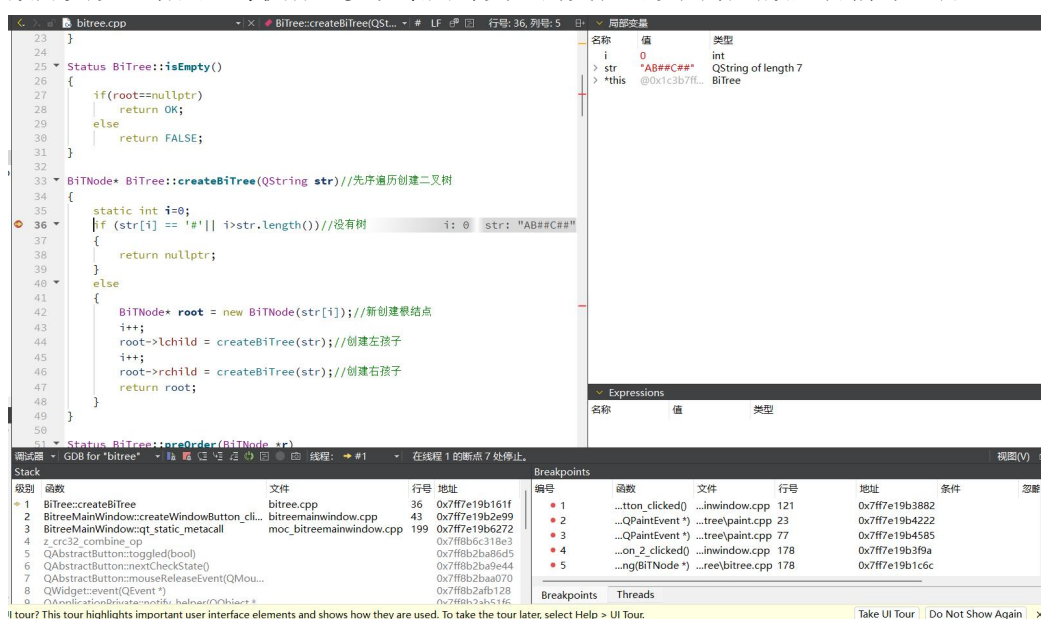
可在上述集成环境下运行;

利用 Qt 自带的 windeployqt 命令, 将 .exe 文件及其依赖项复制到新文件夹, 在该文件夹内点击 .exe 文件即可运行程序, 可在普通 windows 机型下运行。

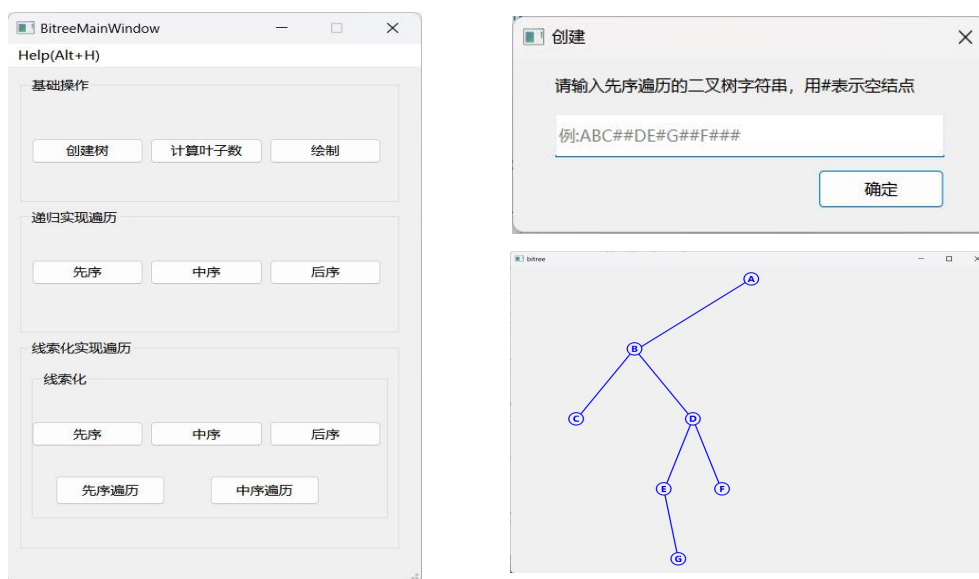
1.6 系统的运行结果分析说明

1.6.1 调试及开发过程

本次开发采用的是 Qt, 在 Qt Creator 中开发调试, 使用 QT 自带的调试器, 通过在函数开头设置断点, 每执行一步观察局部变量的变化可以判断函数是否编写正确。



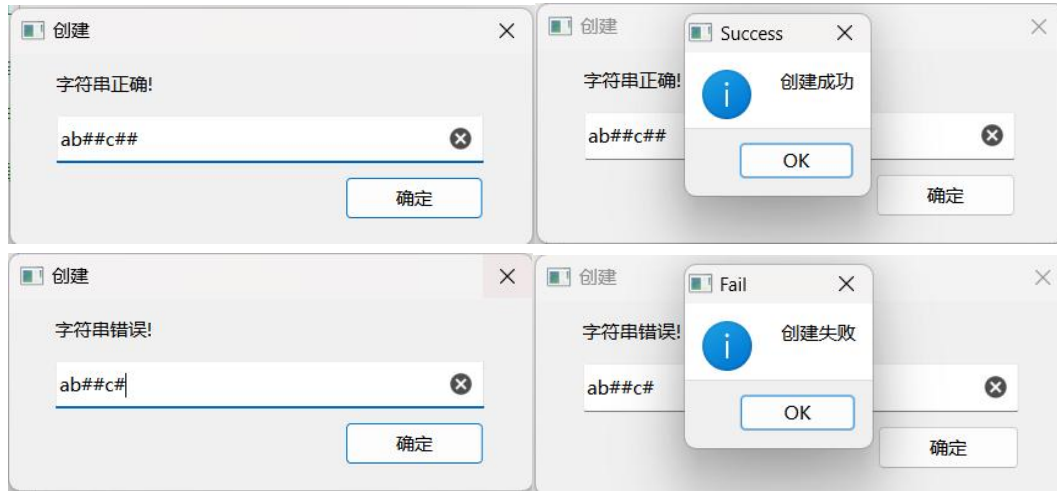
完成二叉树类的所有函数编写后, 开始设计主界面, 在 ui 界面中右键点击组件选择转到槽, 即可在编辑界面编写槽函数代码, 完成后点击构建后再运行 (或者直接运行) 可以查看槽函数效果。下图为设计的三个界面, 分别为主界面、创建界面、绘制界面。



1.6.2 成果

(1) 正确性

1. 创建树时输入字符串能自动检测字符串正确性，同时确保不会根据错误的字符串创建，并且设置输入字符串最大长度为 15。



(2) 稳定性

1. 使用宏定义和自定义符，可以根据需要快速修改

```
#define TRUE      1
#define FALSE    0
#define OK       1
#define ERROR    0
#define INFEASIBLE -1
#define Overflow -2
```

```
typedef int Status;
```

```
typedef QChar ElemType;
```

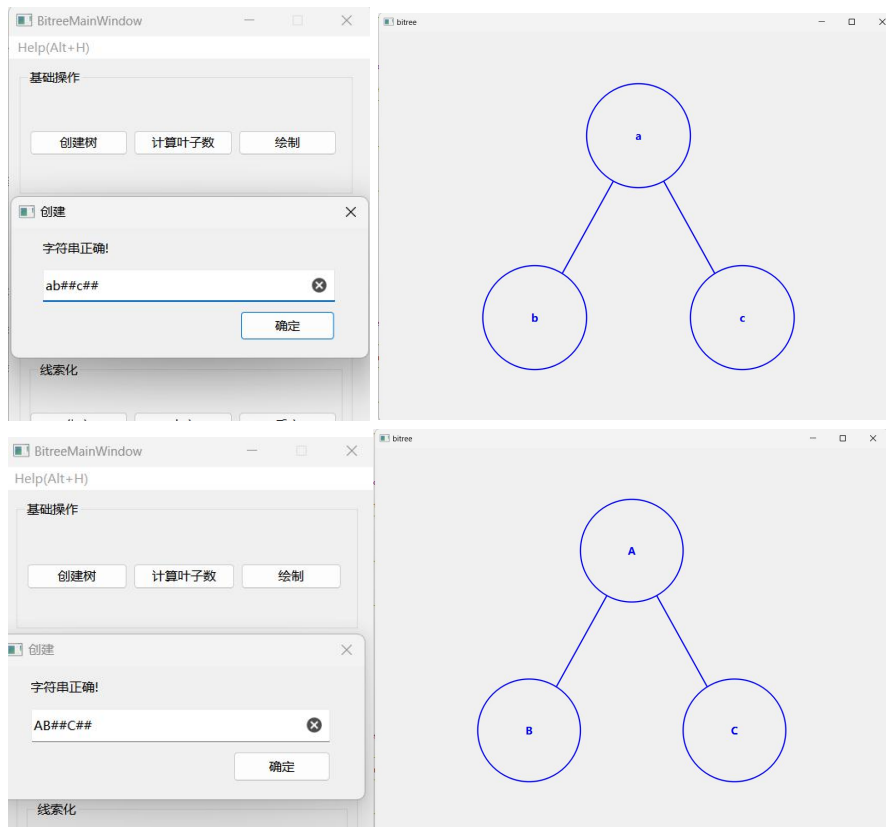
2. 设置 cleanThread 用于清理线索，使同一个树可以进行多次不同的线索化

```
Status BiTree::cleanThread(BiTNode *r){
    // 复原左子树
    if(r==nullptr)return OK;
    if (r->lTag) {
        r->lTag=0;
        r->lchild = nullptr;
    } else {
        cleanThread(r->lchild);
    }
    // 复原右子树
    if (r->rTag) {
        r->rTag=0;
        r->rchild = nullptr;
    } else {
        cleanThread(r->rchild);
    }

    ifThread=false;
    ThreadType=0;
    return OK;
}
```

```
if(tree.ifThread){
    tree.cleanThread(tree.root);
}
tree.postOrderThreading(tree.root);
QMessageBox::information(this, "Success:");
```

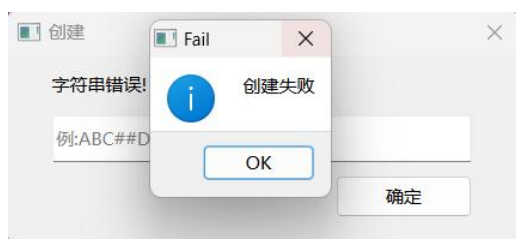
3. 可以重复创建树



(3) 容错性

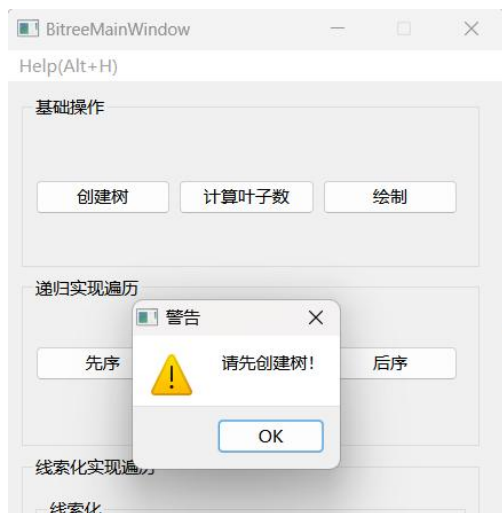
1. 边界条件

空树不能创建

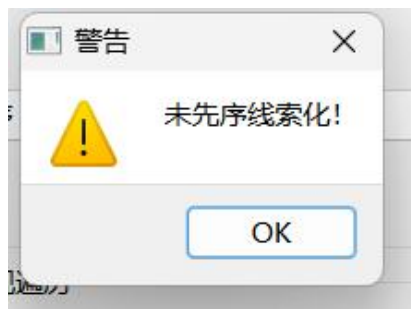


2. 设置警告

创建树之前，其他功能都不能使用



3. 没有创建正确的线索树就不能遍历

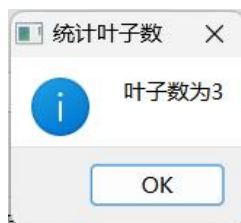


1.7 操作说明

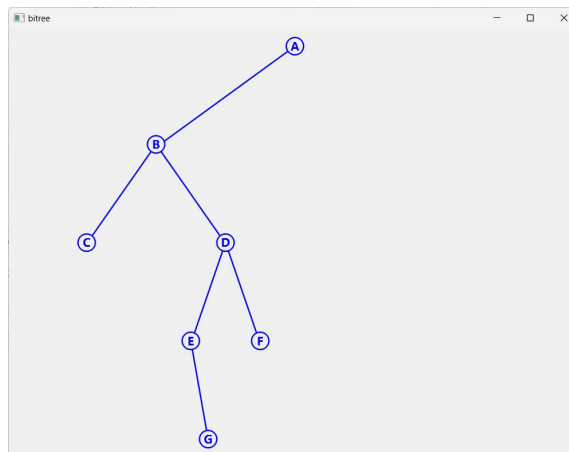
1. 创建树



2. 计算叶子数



3. 显示树



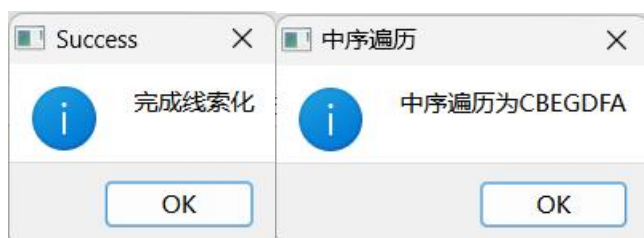
4. 先序、中序、后序遍历



5. 构建先序线索树并遍历



6. 构建中序线索树并遍历



第二部分 综合应用设计说明

2.1 题目

2. ★★★★★上海的地铁交通网络已基本成型，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 地铁网络的图形化显示

用 QGraphicsView 和 QGraphicsScene 来显示地铁网络图，用 Graphics_view_zoom 类来辅助实现地图的缩放。

2.2.2 动态添加地铁线路和站点

通过 ManageLines 类提供用户界面，允许通过 managelines 界面添加和管理地铁线路和站点。

2.2.3 换乘指南算法

实现算法来计算从起点站到终点站的最优换乘路径。

2.2.4 路径显示

在图形界面上动态显示选择的最优换乘路径。

2.3 设计思想

1. 地铁站点和线路的数据模型

使用 Station 和 Line 类来表示地铁站点和线路，SubwayGraph 类用于统计所有线路。

2. 图形化显示地铁网络结构

使用 QGraphicsView 和 QGraphicsScene 来创建一个可视化的地铁网络图。地铁站点使用 QGraphicsEllipseItem 表示, 线路使用 QGraphicsLineItem 表示。通过 MainWindow 类中的 drawStations 和 drawEdges 函数来绘制站点和线路。

3. 动态添加地铁线路和站点

ManageLines 类提供用户界面, 允许用户输入线路和站点的信息。用户输入的信息通过 SubwayGraph 类的 addLine 和 addStation 方法添加到地铁网络中。

4. 换乘指南算法

使用图论中的最短路径算法 (如 Dijkstra 算法) 来计算从起点站到终点站的最优换乘路径。SubwayGraph 类中的 queryTransferMinTime 和 queryTransferMinTransfer 方法用于计算两种不同策略的换乘路径。

5. 显示乘车路径

根据算法计算出的路径, 在图形界面上动态显示。使用不同颜色或标记来突出显示换乘路径。

7. 用户界面设计

使用 Qt Designer 设计用户界面, 包括菜单栏、工具栏、状态栏和对话框等。提供按钮和输入框供用户输入起点站和终点站。显示换乘指南的文本信息和图形路径。

2.4 逻辑结构与物理结构

1. Station 类: 表示地铁站点, 包含站点 ID、名称、经纬度和所属线路等信息。

```
class Station
{
protected:
    int id; //站点ID
    QString name; //站点名称
    double longitude, latitude; //站点经纬度
    QSet<int> linesInfo; //站点所属线路
    static double minLongitude, minLatitude, maxLongitude, maxLatitude; //所有站点的边界位置

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

protected:
    int distance(Station other); //求站点间实地直线距离
    friend class SubwayGraph;
    friend class QTextStream;
};
```

2. Line 类: 表示地铁线路, 包含线路 ID、名称、颜色、站点列表和站点连接关系等信息。

```
typedef QPair<int,int> Edge;
class SubwayGraph;
class QTextStream;
class Line
{
protected:
    int id; //线路ID
    QString name; //线路名称
    QColor color; //线路颜色
    QVector<QString> fromTo; //线路起始站点
    QSet<int> stationsSet; //线路站点集合
    QSet<Edge> edges; //线路站点连接关系

public:
    Line();
    Line(QString lineName, QColor lineColor):name(lineName), color(lineColor)
    {};
    friend class SubwayGraph;
    friend class QTextStream;
};
```

3. SubwayGraph 类：管理整个地铁网络，包含所有站点和线路的集合，以及站点间的连接关系。

```
//图的邻接点结构
class Node{
public:
    int stationID;        //邻接点ID
    double distance;      //两点距离

    Node(){};
    Node(int s, double dist) :stationID(s), distance(dist)
    {};
    //">"运算重载，用于小顶堆
    bool operator > (const Node& n) const
    {
        return this->distance>n.distance;
    }
};

class SubwayGraph{
protected:
    QVector<Station> stations;        //存储所有站点
    QVector<Line> lines;              //存储所有线路
    QHash<QString, int> stationsHash; //站点名到存储位置的hash
    QHash<QString, int> linesHash;    //线路名到存储位置的hash
    QSet<Edge> edges;                //所有边的集合
    QVector<QVector<Node>> graph;     //地铁线路网络图
public:
    SubwayGraph();
    QString getLineName(int l);
    QColor getLineColor(int l);
    int getLineHash(QString lineName);
    QList<int> getLinesHash(QList<QString> linesList);
    QList<QString> getLinesNameList();
    QList<QString> getLineStationsList(int l); //获取线路的所有包含站点
    QString getStationName(int s);
    QPointF getStationCoord(int s);
    QPointF getMinCoord();
    QPointF getMaxCoord();
    QList<int> getStationLinesInfo(int s);
    QList<int> getCommonLines(int s1, int s2); //获取两个站点的公共所属线路
    int getStationHash(QString stationName);
    QList<QString> getStationsNameList();

    void addLine(QString lineName, QColor color); //添加新线路
    void addStation(Station s); //添加新站点
    void addConnection(int s1, int s2, int l); //添加站点连接关系
    void getGraph(QList<int>&stationsList, QList<Edge>&edgesList);
    //获取最少时间的线路
    bool queryTransferMinTime(int s1, int s2, QList<int>&stationsList, QList<Edge>&edgesList);
    //获取最少换乘的线路
    bool queryTransferMinTransfer(int s1, int s2, QList<int>&stationsList, QList<Edge>&edgesList);
    bool readFileData(QString fileName);
private:
    void clearData();
    bool insertEdge(int s1, int s2);
    void updateMinMaxLongilati();
    void makeGraph();
};
```

2.5 开发平台

2.5.1 开发平台

开发语言：C++（C++11 标准以上）

开发框架：QT

集成开发环境：Qt Version 6.5.3

编译器：MinGW 11.2.0 64-bit for c++

2.5.2 运行环境

可在上述集成环境下运行；

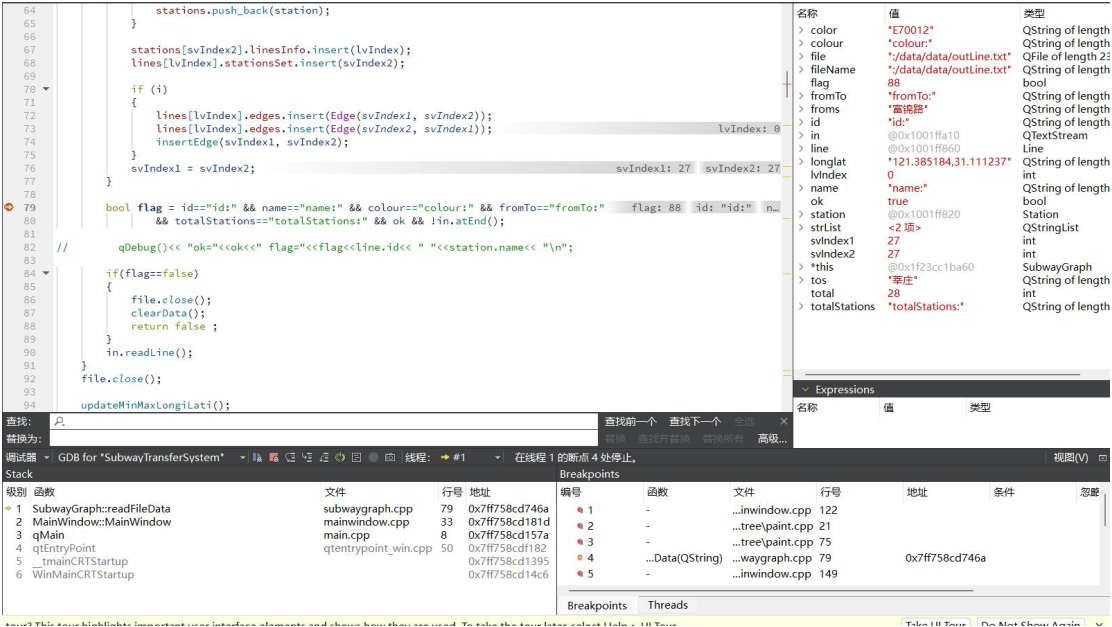
利用 Qt 自带的 windeployqt 命令，将.exe 文件及其依赖项复制到新文件夹，在该文件

夹内点击.exe 文件即可运行程序，可在普通 windows 机型下运行。

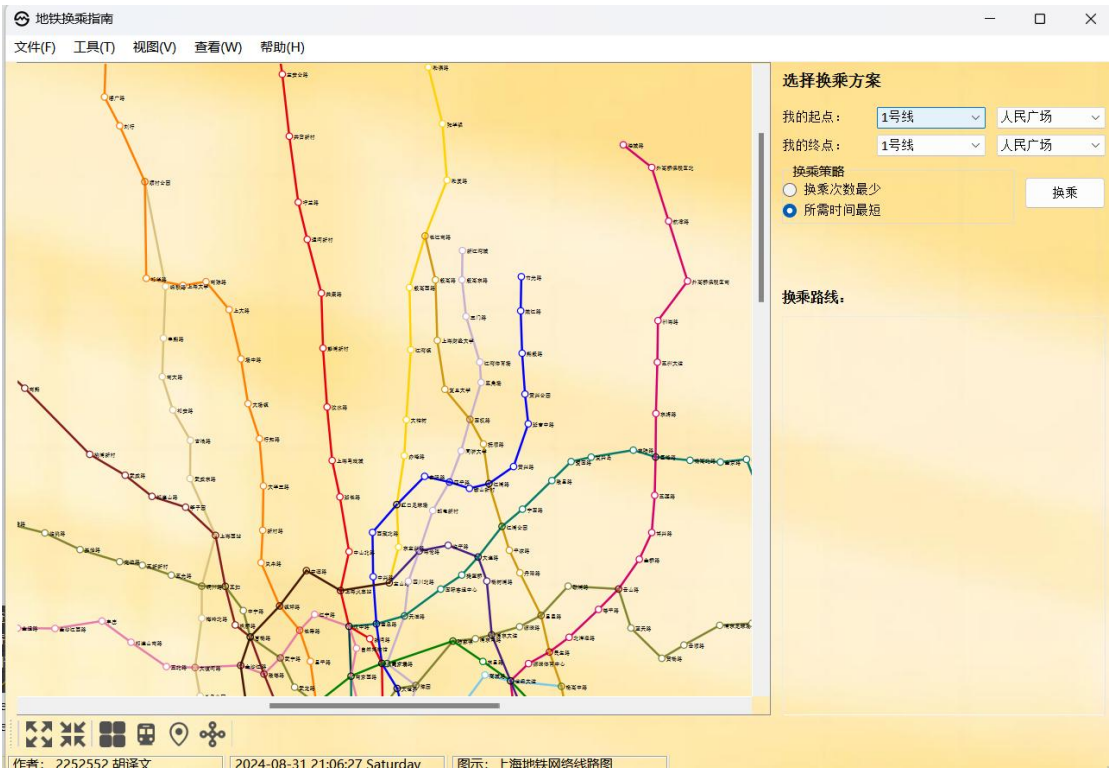
2.6 系统的运行结果分析说明

1.6.1 调试及开发过程

本次开发采用的是 Qt，在 Qt Creator 中开发调试，使用 QT 自带的调试器，通过在函数开头设置断点，每执行一步观察局部变量的变化可以判断函数是否编写正确。



完成类的所有函数编写后，开始设计主界面，在 ui 界面中右键点击组件选择转到槽，即可在编辑界面编写槽函数代码，完成后点击构建后再运行（或者直接运行）可以查看槽函数效果。下图为设计的两个界面，分别为主界面、添加界面。





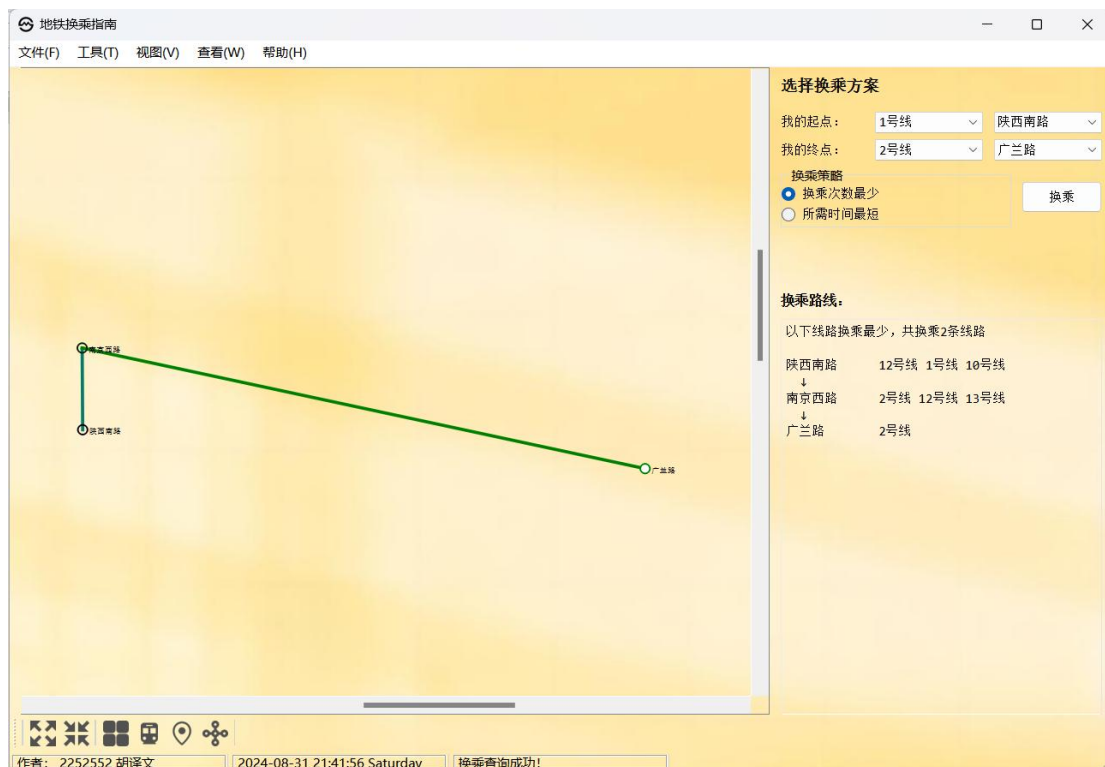
1.6.2 成果

(1) 正确性

经过多次验证，程序运行与预期结果相符。

如下换乘方案:

时间最短策略



换乘次数最少策略



(2) 稳定性

1. 图形界面的稳定性:

利用 Qt 框架的成熟组件, 如 QGraphicsView 和 QGraphicsScene, 这些组件经过广泛测试, 具有很高的稳定性和性能。通过事件驱动模型处理用户交互, 确保了界面响应的及时性和准确性。无论是在放大窗口或是小窗口程序都未有任何异常, 视图和右侧栏会随窗口大小自动调节

2. 算法的可靠性:

使用经典的图论算法 (如 Dijkstra 算法) 来计算换乘路径, 这些算法在理论上和实践上都经过了严格的验证。算法实现考虑了各种边界条件和异常情况, 如起点和终点相同、无可行路径等, 确保了算法的鲁棒性。运行速度快, 效率高。

3. 数据持久化:

系统能够将地铁网络数据保存到文件中, 并在需要时重新加载, 这样即使在系统重启后也能恢复到之前的状态。数据的持久化也方便了数据的备份和恢复, 增强了系统的稳定性。

(4) 容错性

1. 非法操作提示

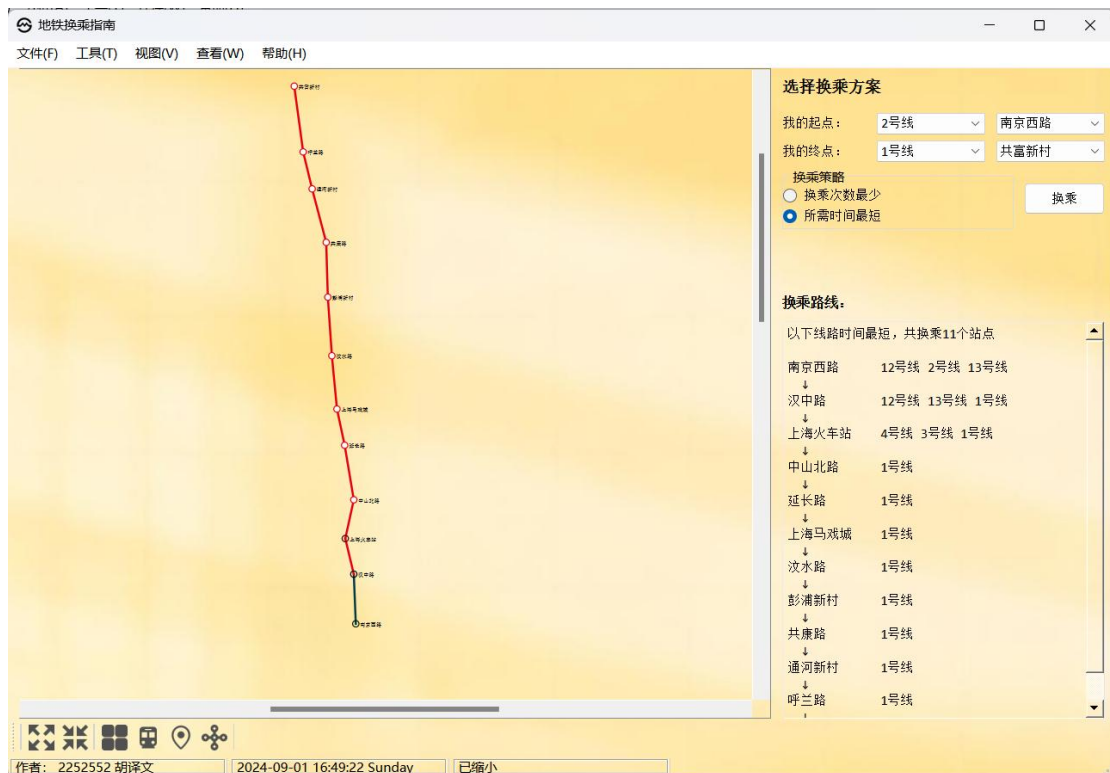


2.7 操作说明

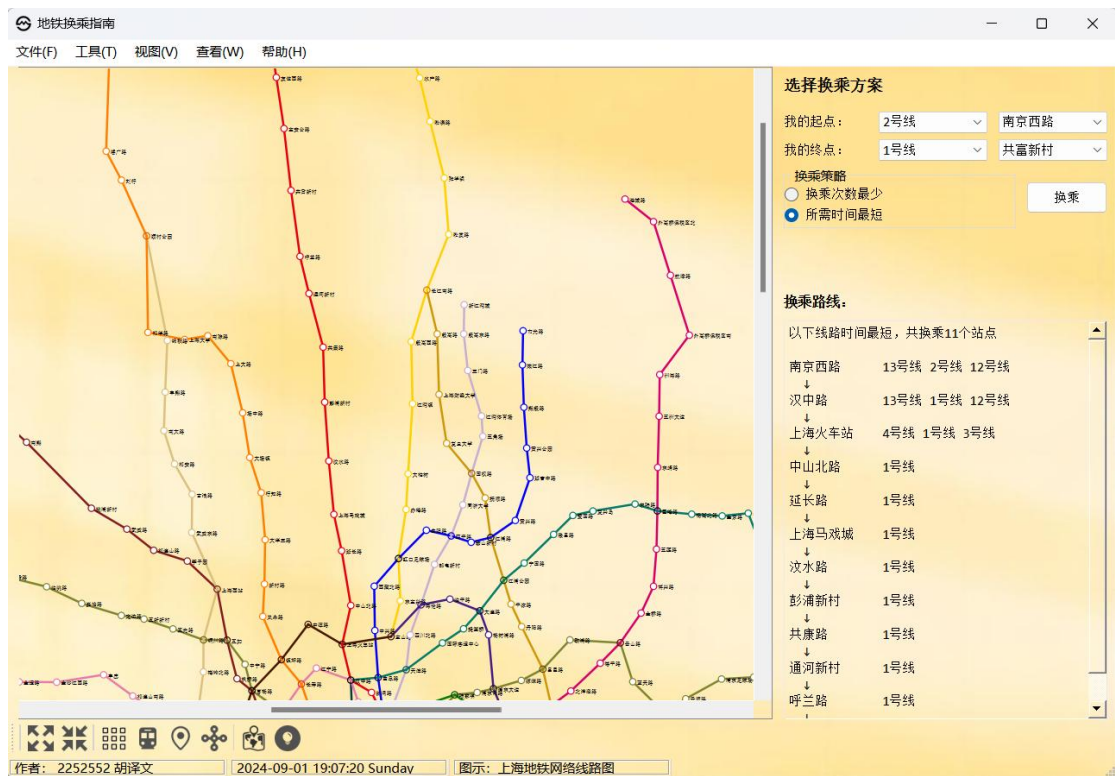
1. 搜索换乘方案

选择起点与终点所在线路，站点列表会自动变成所选线路的站点，此时可以选择站点。选择起点、终点完成后，再选择一个换乘策略，最后点击换乘按钮，右下角即可看见文字版换乘方案，绘制区域出现换乘线路图。

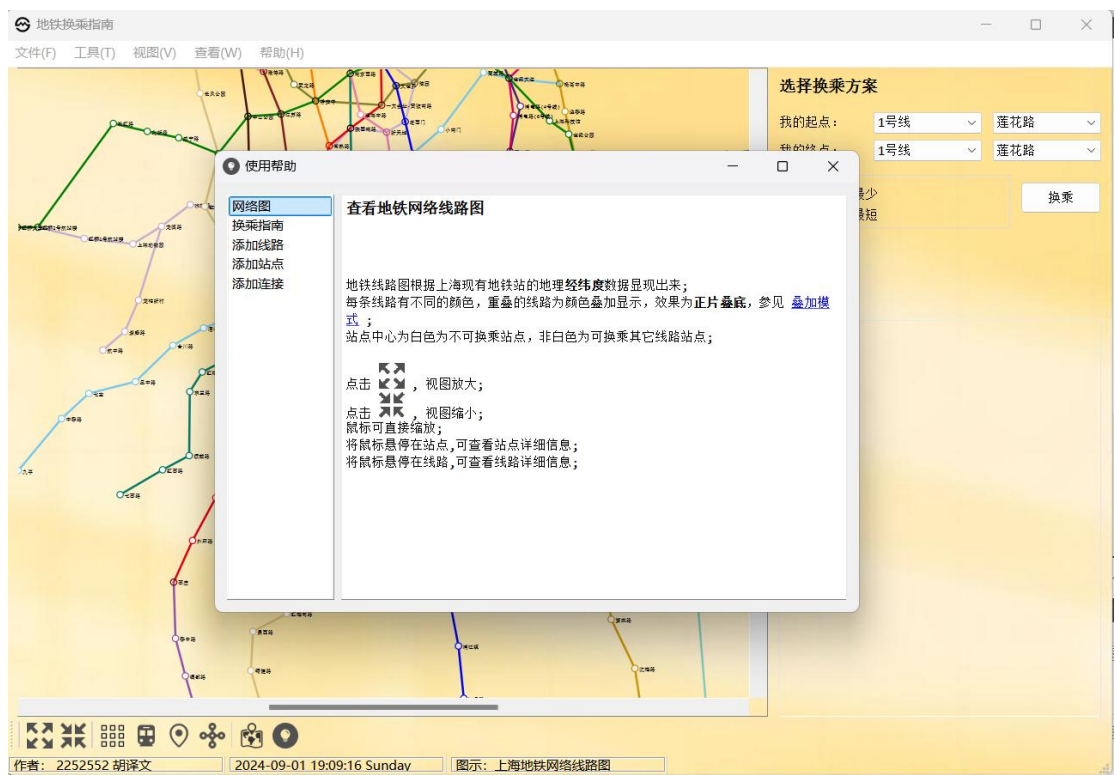





2. 点击工具栏中的  , 可以显示所有线路图



3. 点击工具栏中的 ，可以显示使用帮助



4. 点击工具栏中的 ，可以打开所有与添加线路或站点有关的窗口，或者点击



线路

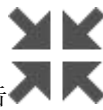
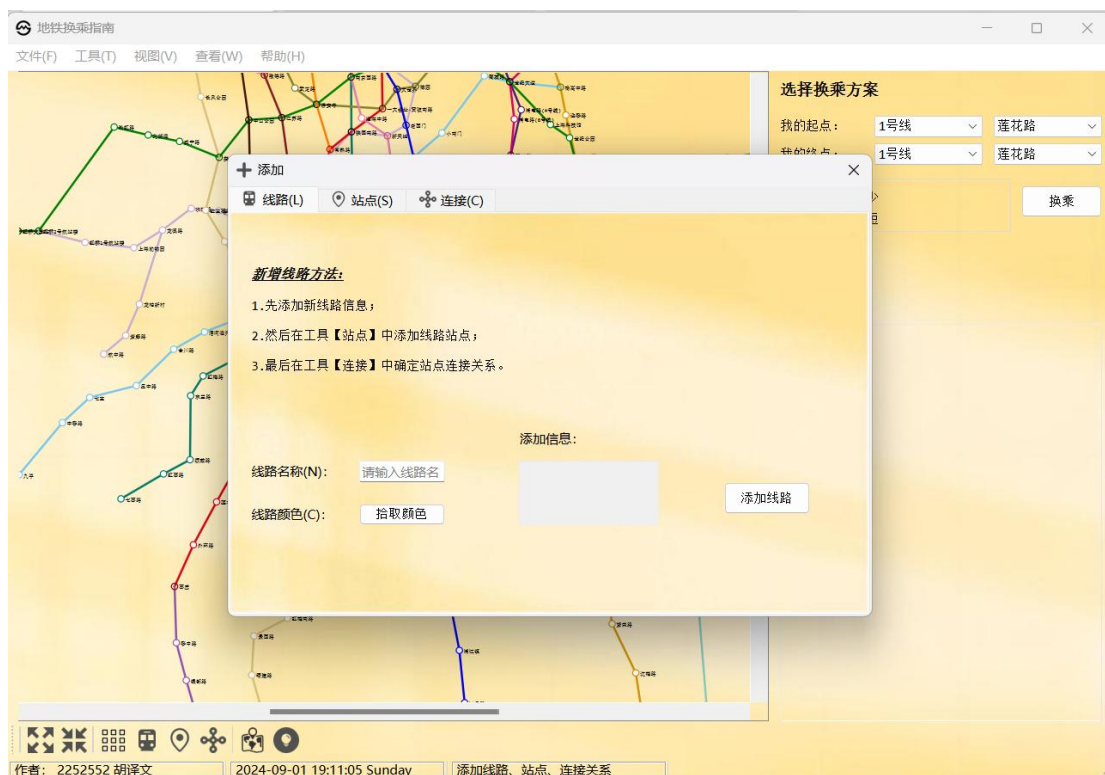


站点



连接都可以打开对应窗口，按照窗口提示可以添加线路或

站点



5. 对于线路图区域，点击可放大，点击可缩小，也可用鼠标操作



第三部分 实践总结

3.1. 所做的工作

在本次课程设计中，我学习了使用 Qt 框架进行跨平台 GUI 应用程序的开发以及如何设计稳定可靠的系统，包括异常处理和错误恢复机制。理解了模块化设计的重要性，并将其应用于系统开发中。还学习了软件工程的基本原则，如代码的可维护性、可扩展性和重用性。

3.2. 总结与收获

通过这些工作，我不仅提升了编程技能，还学习了软件设计、数据结构、算法实现和系统开发等多方面的知识。这些经验对于未来解决更复杂的问题和开发更大型的软件系统都是非常宝贵的。

第四部分 参考文献

- [1] QT5 基础教程（介绍，下载，安装，第一个 QT 程序）
https://blog.csdn.net/qq_41854911/article/details/122050335
- [2] Qt5 开发从入门到精通——第一篇概述
https://blog.csdn.net/2401_85013863/article/details/138988281
- [3] 霍亚飞. Qt Creator 快速入门第二版[M]. 北京：北京航空航天大学出版社, 2014-1
- [4] 霍亚飞，程梁. QT5 编程入门[M]. 北京：北京航空航天大学出版社, 2015-1