《数据结构》课程设计总结

学号：_____

姓名：_____

专业：__

目　录

第一部分 算法实现设计说明

1.1 题目

以邻接矩阵的方式确定一个图，完成：

(1)建立并显示出它的邻接链表；

(2)以递归及非递归的方式进行深度优先遍历，显示遍历的结果，并随时显示栈的入、出情况；

(3)对该图进行广度优先遍历，显示遍历的结果，并随时显示队列的入、出情况。

1.2 软件功能

(1)以邻接矩阵的方式确定一个图并显示出它的邻接链表

实现方式：

1）通过下面的槽函数打开新建图的页面；

```
    //以邻接矩阵的方式确定一个图
void MainWindow::on_actioncreate_triggered()
{
    statuslabel1->setText(tr("以邻接矩阵的方式确定一个图"));
    ui->textBrowser->clear();
    ui->textBrowser_2->clear();
    ui->textBrowser_3->clear();
    buildmap->ui->textEdit->clear();
    buildmap->show();
}
```

2）新建图页面点击按钮，连接到槽函数 addbytext；

```
connect(buildmap->ui->pushButton,&QPushButton::clicked,this,&MainWindow::addbytext);
//建立图
void MainWindow::addbytext()
{
    QMessageBox box;
    box.setWindowTitle(tr("建图"));
    QString filename="addtext.txt";
    QFile file(filename);
    if(!file.open(QIODevice::WriteOnly|QIODevice::Text))
    {
        return;
```

```
    }
    QTextStream out(&file);
    out<<buildmap->ui->textEdit->toPlainText();
    file.close();
    bool flag = manager->readfile(filename);
    if(flag)
    {
        box.setIcon(QMessageBox::Information);
        box.setText(tr("添加成功"));
        manager->toadjlist();
        adjlistupdate();
    }
    else
    {
        box.setIcon(QMessageBox::Warning);
        box.setText(tr("添加失败"));
    }
    box.addButton(tr("确定"),QMessageBox::AcceptRole);
    if(box.exec()==QMessageBox::Accepted)
    {
        box.close();
    }
}
```

3）该槽函数调用 readfile 函数，完成数据读取，调用 toadjlist 函数完成邻接链表建立，调用 adjlistupdate 函数实现邻接链表显示；

```
//文件读取
bool Manager::readfile(QString filename)
{
    memset(adj,0,sizeof(adj));
    for(int i=0;i<n;i++)
    {
        link *t=&adjl[i];
        link *p;
        while(t!=NULL)
        {
            p=t;
            t=t->next;
            delete p;
        }
    }
    QFile file(filename);
    if(!file.open(QIODevice::ReadOnly))
        return false;
```

```
    QTextStream in(&file);
    while(!in.atEnd())
    {
        in>>n;
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                in>>adj[i][j];
            }
        }
    }
    file.close();
    if(n==0)
        return false;
    return true;
}
//由邻接矩阵转为邻接链表
void Manager::toadjlist()
{
    for (int i = 0; i < n; i++)
    {
        link *t = &adjl[i];
        t->id = i;
        for (int j = 0; j < n; j++)
        {
            if (true == adj[i][j])
            {
                link* temp=new link;
                temp->id = j;
                temp->next = NULL;
                t->next = temp;
                t = t->next;
            }
        }
    }
}
//更新邻接链表
void MainWindow::adjlistupdate()
{
    statuslabel1->setText(tr("邻接链表已更新"));
    QString text="邻接链表如下:\n\n";
    for(int i=0; i<manager->n; i++)
```

```
    {
        link* temp = &manager->adjl[i];
        while(temp!=NULL)
        {
            text+=QString::number(temp->id);
            if(temp->next!=NULL)
                text+="->";
            temp=temp->next;
        }
        text+="\n\n";
    }
    ui->textBrowser->clear();
    ui->textBrowser->setText(text);
}
```

(2)以递归方式进行深度优先遍历，显示遍历的结果

实现方式：

1）点击按钮调用下面的槽函数；

```
//递归方式深度优先遍历
void MainWindow::on_actiondfsnostack_triggered()
{
    if(false==check())
        return;
    statuslabel1->setText(tr("递归方式深度优先遍历"));
    ui->textBrowser_2->clear();
    memset(manager->visit, 0, sizeof(manager->visit));
    QString text, plus;
    for (int i = 0; i < manager->n; i++)
    {
        if (manager->visit[i] == 0)
        {
            text+=QString::number(i);
            manager->visit[i] = 1;
            manager->dfs_nostack(i, plus);
            text+=plus;
            plus.clear();
            text+="\n\n";
        }
    }
    ui->textBrowser_2->setText(text);
}
```

2）上述函数调用 check 函数来确保已经建立图，调用 dfs_nostack 函数进行深度优先遍历；

```
//查看是否可以遍历
bool MainWindow::check()
{
    if(manager->n==0)
    {
        QMessageBox box;
        box.setWindowTitle("错误");
        box.setIcon(QMessageBox::Warning);
        box.setText("还未建立图");
        box.addButton("确定",QMessageBox::AcceptRole);
        if(box.exec() == QMessageBox::Accepted)
        {
            box.close();
        }
        return false;
    }
    return true;
}
//递归方式深度优先遍历
void Manager::dfs_nostack(int id, QString &plus)
{
    link* p = &adjl[id];
    while (p != NULL)
    {
        if (p != NULL && visit[p->id] == 0)
        {
            visit[p->id] = 1;
            plus+="->";
            plus+=QString::number(p->id);
            dfs_nostack(p->id, plus);
        }
        else
            p = p->next;
    }
}
```

(3)以非递归方式进行深度优先遍历，显示遍历的结果，并随时显示栈的入、出情况

实现方式：

1）点击按钮调用下面的槽函数；

```
//非递归方式深度优先遍历
void MainWindow::on_actiondfsstack_triggered()
{
    if(false==check())
        return;
    statuslabel1->setText(tr("非递归方式深度优先遍历"));
    ui->textBrowser_2->clear();
    memset(manager->visit, 0, sizeof(manager->visit));
    manager->show.clear();
    manager->s.clear();
    QString text, plus;
    for (int i = 0; i < manager->n; i++)
    {
        if (manager->visit[i] == 0)
        {
            text+=QString::number(i);
            manager->visit[i] = 1;
            manager->dfs_stack(i, plus);
            text+=plus;
            plus.clear();
            text+="\n\n";
        }
    }
    qDebug()<<manager->show;
    for (int i = 0; i < manager->show.size(); i++)
    {
        QThread::msleep(1000);
        if(manager->show.at(i)=='i')
        {
            stackflag++;
            stackbutton[stackflag-1]->setText(QString(manager->show[i-1]));
            stackbutton[stackflag-1]->setVisible(true);
            QCoreApplication::processEvents();
        }
        else if(manager->show.at(i)=='o')
        {
            stackflag--;
            stackbutton[stackflag]->setVisible(false);
            QCoreApplication::processEvents();
        }
    }
    ui->textBrowser_2->setText(text);
}
```

2）上述函数调用 check 函数来确保已经建立图，调用 dfs_stack 函数进行深度优先遍历；

```
//非递归方式深度优先遍历
void Manager::dfs_stack(int id, QString &plus)
{
    link* p = adjl[id].next;
    while(p != NULL)
    {
        s.push(p->id);
        show+=QString::number(p->id);
        show+='i';
        p = p->next;
    }
    while (!s.empty())
    {
        int temp = s.top();
        if (visit[temp] == 0)
        {
            plus+="->";
            plus+=QString::number(temp);
            visit[temp] = 1;
        }
        s.pop();
        show+=QString::number(temp);
        show+='o';
        link* t = adjl[temp].next;
        while (t != NULL)
        {
            if (visit[t->id] == 0)
            {
                s.push(t->id);
                show+=QString::number(t->id);
                show+='i';
                t = t->next;
            }
            else
                t = t->next;
        }
    }
}
```

(4)进行广度优先遍历，显示遍历的结果，并随时显示队列的入、出情况

实现方式：

1）点击按钮调用下面的槽函数；

```
//广度优先遍历
void MainWindow::on_actionbfs_triggered()
{
    if(false==check())
        return;
    statuslabel1->setText(tr("广度优先遍历"));
    ui->textBrowser_3->clear();
    memset(manager->visit, 0, sizeof(manager->visit));
    manager->show.clear();
    manager->q.clear();
    QString text, plus;
    for (int i = 0; i < manager->n; ++i)
    {
        if (manager->visit[i] == 0)
        {
            text+=QString::number(i);
            manager->visit[i] = 1;
            manager->bfs(i, plus);
            text+=plus;
            plus.clear();
            text+="\n\n";
        }
    }
    qDebug()<<manager->show;
    for (int i = 0; i < manager->show.size(); ++i)
    {
        QThread::msleep(1000);
        if(manager->show.at(i)=='i')
        {
            queueflag++;
            queuebutton[queueflag-1]->setText(QString(manager->show[i-1]));
            queuebutton[queueflag-1]->setVisible(true);
            QCoreApplication::processEvents();
        }
        else if(manager->show.at(i)=='o')
        {
            queueflag--;
            for(int j=0;j<queueflag;j++)
            {
                QString str= queuebutton[j+1]->text();
                queuebutton[j]->setText(str);
            }
            queuebutton[queueflag]->setVisible(false);
            QCoreApplication::processEvents();
```

```
        }
    }
    ui->textBrowser_3->setText(text);
}
```

2）上述函数调用 check 函数来确保已经建立图，调用 bfs 函数进行广度优先遍历；

```
//广度优先遍历
void Manager::bfs(int id, QString &plus)
{
    link* p = adjl[id].next;
    while (p != NULL)
    {
        q.enqueue(p->id);
        show+=QString::number(p->id);
        show+='i';
        p = p->next;
    }
    while (!q.empty())
    {
        int temp = q.front();
        q.dequeue();
        show+=QString::number(temp);
        show+='o';
        if (visit[temp] == 0)
        {
            plus+="->";
            plus+=QString::number(temp);
            visit[temp] = 1;
        }
        link* t = adjl[temp].next;
        while (t != NULL)
        {
            if (visit[t->id] == 0)
            {
                q.enqueue(t->id);
                show+=QString::number(t->id);
                show+='i';
                t = t->next;
            }
            else
                t = t->next;
        }
    }
}
```

(5)查看使用帮助

实现方式：

1）点击按钮调用下面的槽函数；

```
//使用帮助
void MainWindow::on_actionhelp_triggered()
{
    usehelp->show();
}
```

2）usehelp 为通过 ui 设计实现的窗口；

(6)查看关于 Qt 和关于作者

实现方式：

1）点击按钮调用下面的槽函数；

```
//关于 Qt
void MainWindow::on_actionaboutqt_triggered()
{
    QMessageBox::aboutQt(this,tr("关于 Qt"));
}
//关于作者
void MainWindow::on_actionaboutauther_triggered()
{
    QMessageBox box;
    box.setWindowTitle(tr("关于作者"));
    box.setIcon(QMessageBox::Information);
    box.setText(tr("学号:1853862\n"
                   "专业:计算机科学与技术\n"
                   "联系方式:841713301@qq.com\n"));
    box.addButton(tr("确定"),QMessageBox::AcceptRole);
    if(box.exec() == QMessageBox::Accepted)
        box.close();
}
```

1.3 设计思想

(1)实现思路

1）分析题目要求，进行宏观设计

首先对题目要求进行分析，看所需实现的功能有哪些，所需要的页面有哪些，页面如何

布局，数据结构如何选择等，有一个较好的整体设计，然后再考虑细化的问题。

2）前后端分离

软件开发过程采用前后端分离。将数据的处理、逻辑的实现等放到后端；将页面的显示，用户的交互等放到前端；前后端通过接口连接或者前端主导后端数据的调用，形成整体的架构。

3）后端实现

软件开发从后端入手，利用之前熟悉的 VS、VSCode 等控制台应用开发工具，将后端实现，在控制台查看数据结构是否合适、算法设计是否合理、最终结果是否符合预期。

4）前端实现

在后端设计完成后，对于前端的每个页面进行分别的 UI 设计，然后逐步实现每一个功能，将前后端对接起来，完成整个程序。

(2)算法设计基本流程

该软件算法较为简单，每个算法部分的耦合性不强，只需要注意一下每个算法部分的前后顺序即可，该软件共有一下几个算法。

1）文件读取

2）由邻接矩阵转邻接链表

3）递归方式深度优先遍历

4）非递归方式深度优先遍历

5）广度优先遍历

每个算法依次进行设计。通过算法 1 获取邻接矩阵，算法 2 获取邻接链表，算法 3、4、5 利用邻接链表完成遍历。

1.4 逻辑结构与物理结构

(1)逻辑结构

本软件主要采用的逻辑结构是线性结构和图形结构；

```
struct link
{
    int id;
    link* next;
```

```
};
int adj[100][100];  //邻接矩阵
link adjl[100];     //邻接链表
bool visit[100];    //是否访问过
QStack<int> s;      //栈
QQueue<int> q;      //队列
```

(2)物理结构

本软件主要采用的物理结构有顺序结构和链式结构，如上面代码所示。

1.5 开发平台

(1)开发平台

机器型号：联想 Y7000(2018 款)

CPU：Intel Core i5 8300H

内存：8GB

操作系统：Windows 10

IDE：Qt Creator 4.3.0

开发语言：C++

开发框架：QT 5.9.0

编译器：MinGW 32bit

(2)运行环境

所提交的源代码可在上述 IDE 运行。

所提交的 EXE 文件可在任意 Windows 机型下运行。

1.6 系统的运行结果分析说明

(1)调试及开发过程

1）调试过程：

本软件主要采用 Qt Creator 进行开发调试，由于本软件涉及到大量图形界面和用户交互，所以断点调试、单步运行等方式并不适合。同时因为本软件的复杂逻辑较少，所以在调

试过程中主要使用 qDebug 进行控制台输出完成调试。例如：

```
237  //广度优先遍历
238  void MainWindow::on_actionbfs_triggered()
239  {
240      if(false==check())
241          return;
242      statuslabel1->setText(tr("广度优先遍历"));
243      ui->textBrowser_3->clear();
244      memset(manager->visit, 0, sizeof(manager->visit));
245      manager->show.clear();
246      manager->q.clear();
247      QString text, plus;
248      for (int i = 0; i < manager->n; ++i)
249      {
250          if (manager->visit[i] == 0)
251          {
252              text+=QString::number(i);
253              manager->visit[i] = 1;
254              manager->bfs(i, plus);
255              text+=plus;
256              plus.clear();
257              text+="\n\n";
258          }
259      }
260      qDebug()<<manager->show;
261      for (int i = 0; i < manager->show.size(); ++i)
```

应用程序输出

algorithm

```
Starting D:\programing\myQt\build-algorithm-Desktop_Qt_5_9_0_MinGW_32
"2i3i2o3i3o1i3o1i1o1o"
```

在控制台输出的是 manager->show，这是广度优先遍历队列变化情况，数字代表节点，i 代表入队，o 代表出队。

2）开发过程：

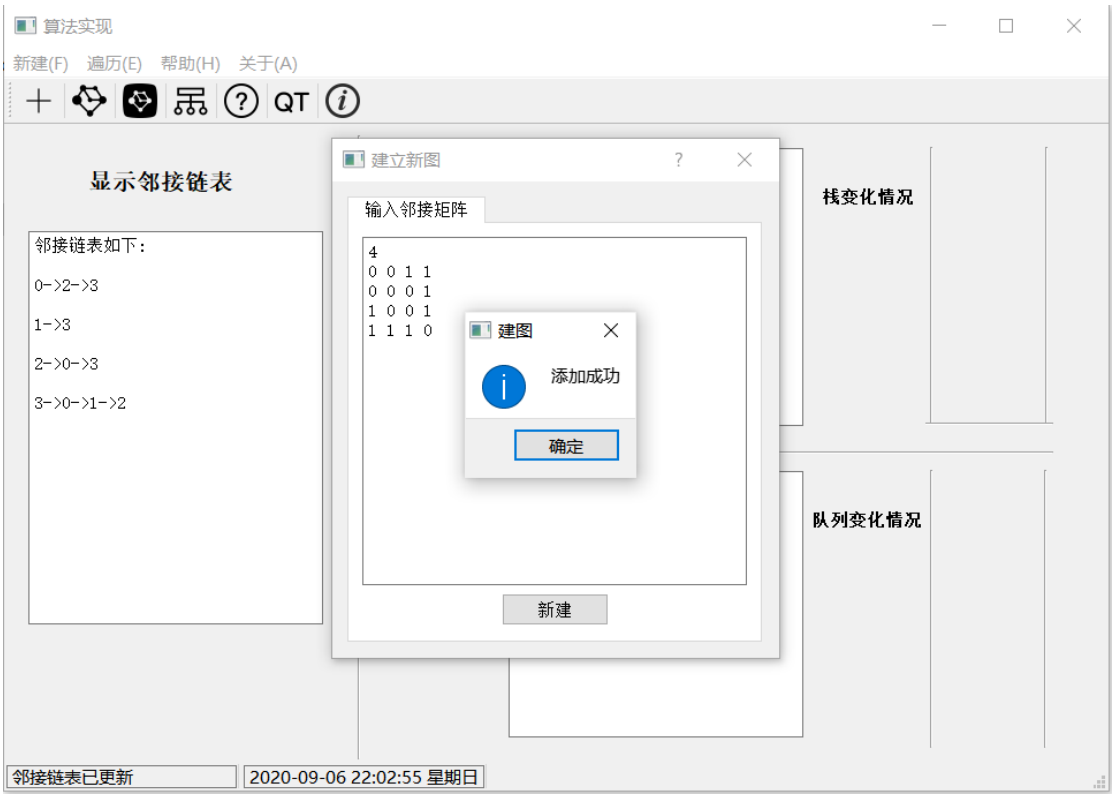
本软件先开发后端，在完成后端后再开发前端。前端根据页面设计，分别开发主页面、新建图页面、使用帮助页面。然后将前后端进行对接，并对每个模块进行开发测试，调整错误，细化代码逻辑，最终完成全部开发。

(2)开发成果

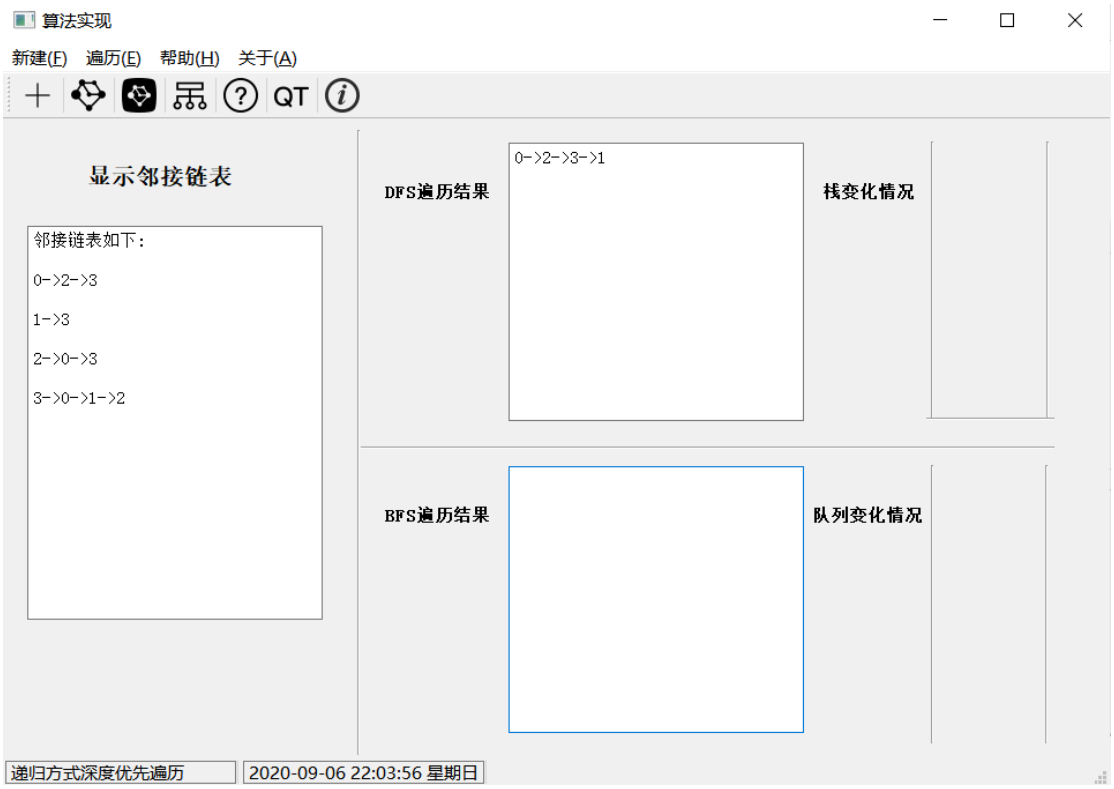
1）正确性：

该软件的每个功能都保证了正确性。

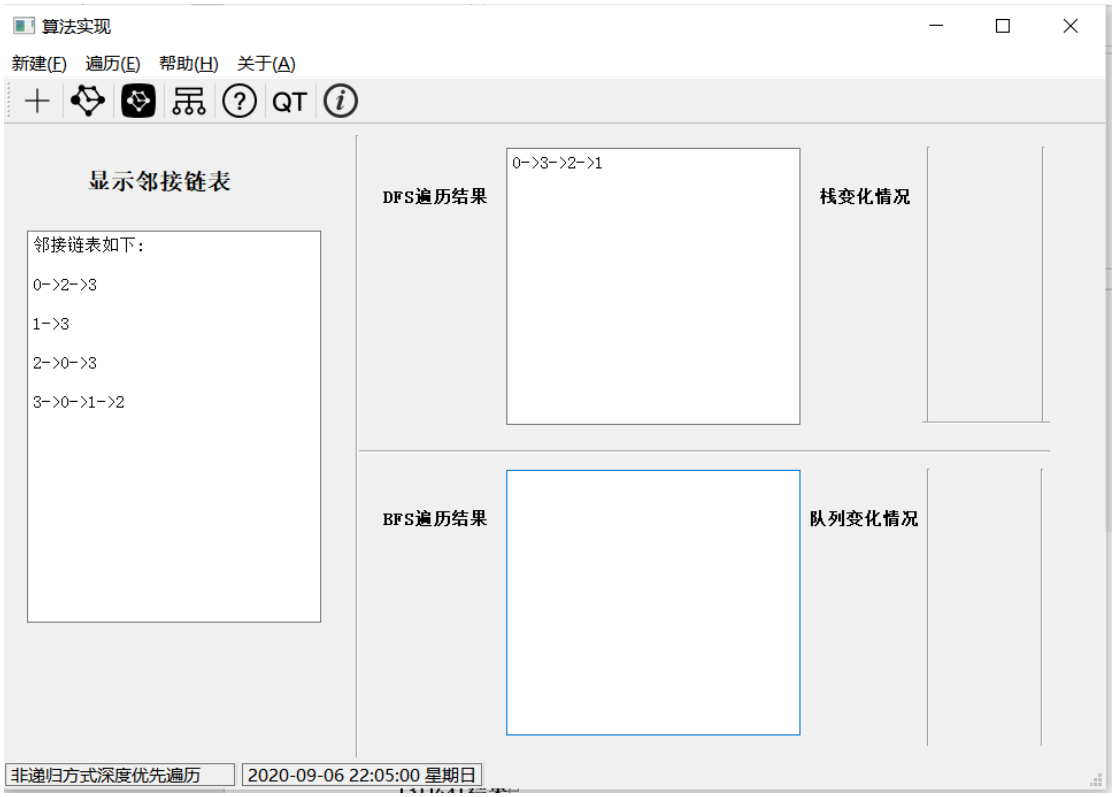
新建图并显示邻接链表：



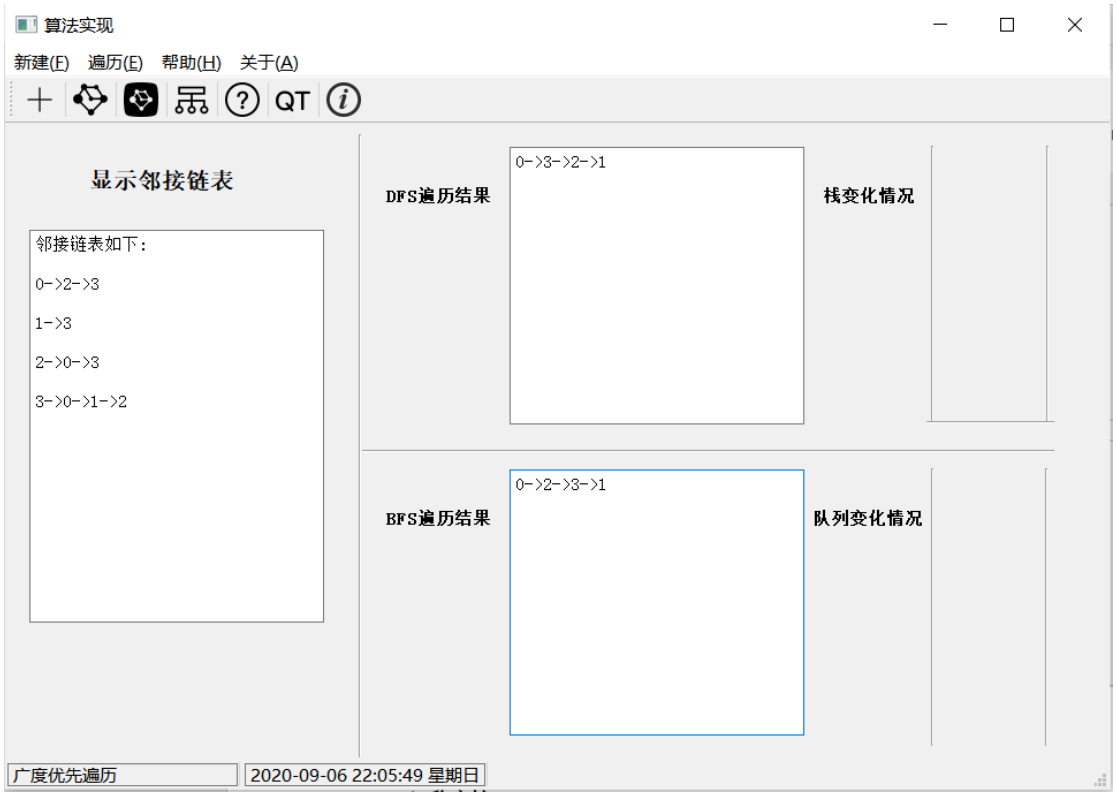
递归方式深度优先遍历：



非递归方式深度优先遍历：



广度优先遍历：



2）稳定性：

程序稳定性良好。一方面程序运行速度正常，所有操作都得到迅速响应，而且不会出现

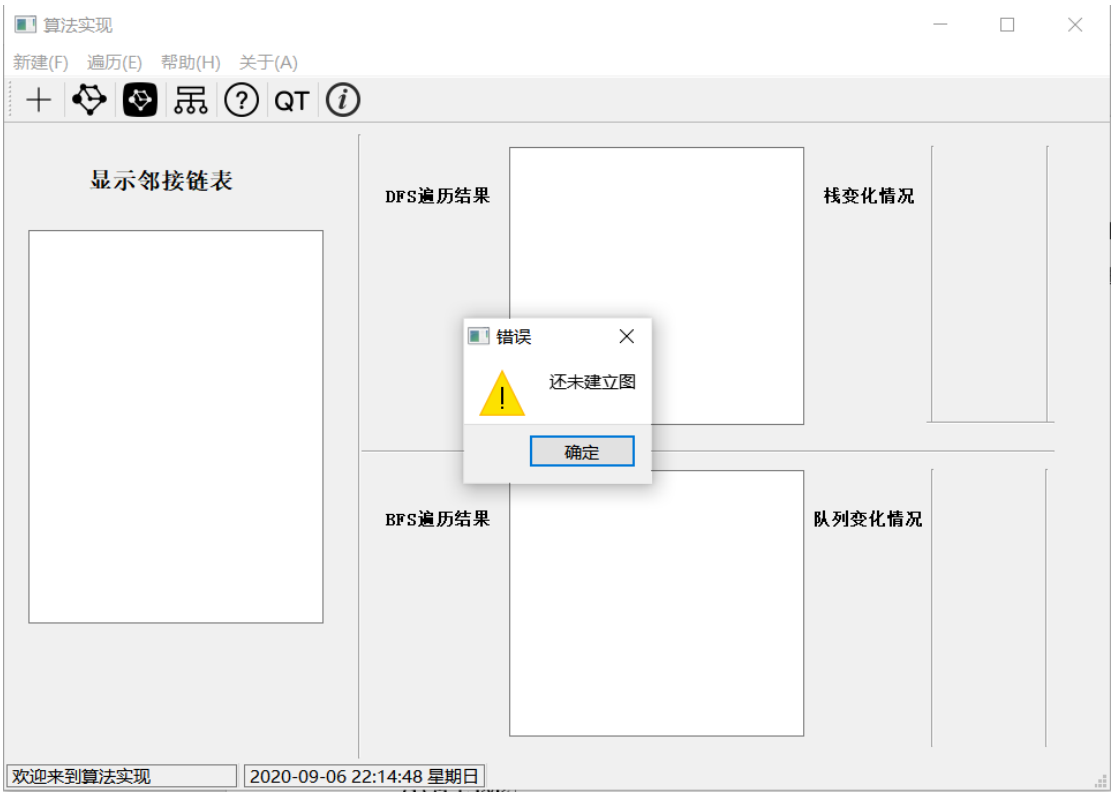
程序卡住未响应的情况；另一方面程序可以稳定多次的新建图，显示邻接链表，进行遍历操作；再一方面，新建图后数据会正常更新。

经过多次全分支覆盖的测试，程序未出现异常。

3）容错能力：

程序容错能力良好。

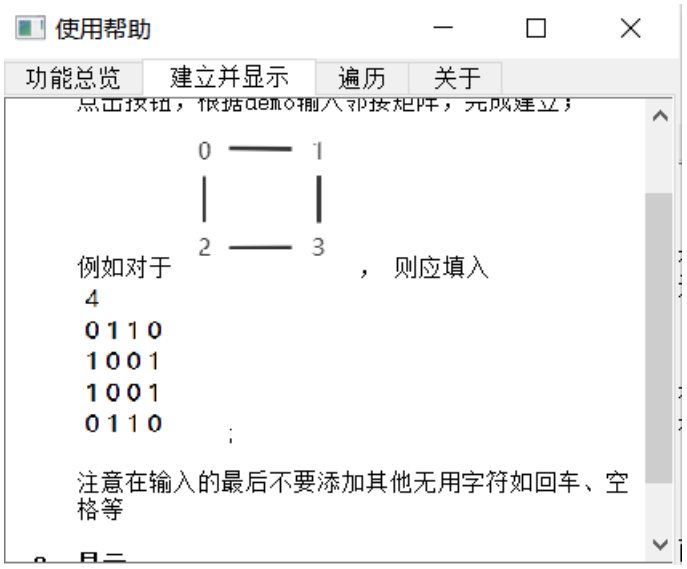
当没有建图时，无法进行遍历；

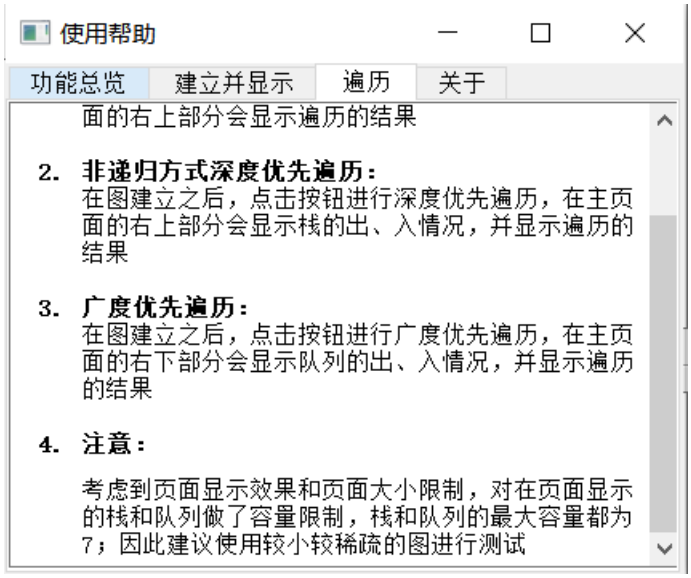


新建图不输入正确格式无法完成新建；

再次新建会抹除之前的数据；

关于程序限制都写入了使用帮助；





(3)运行结果

运行案例请见下方操作说明。

1.7 操作说明

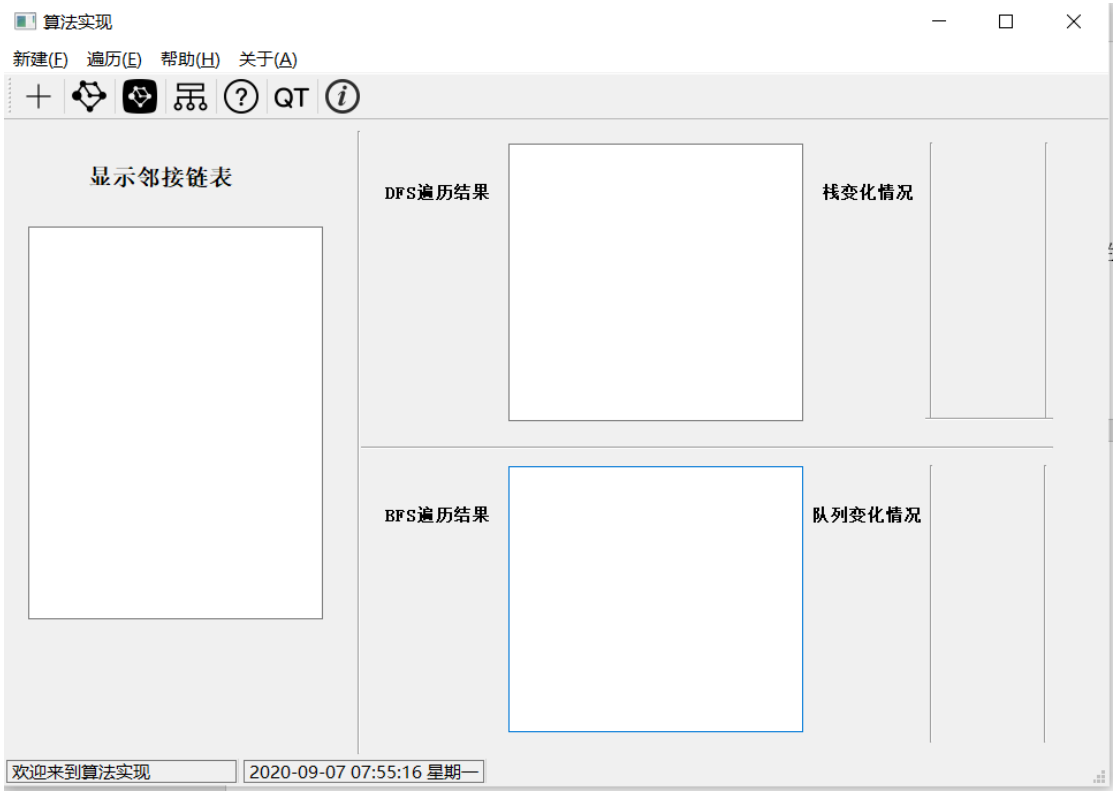
(1)点击 EXE

点击 algprithm_boxed.exe 运行程序，进入主界面。

(2)主界面

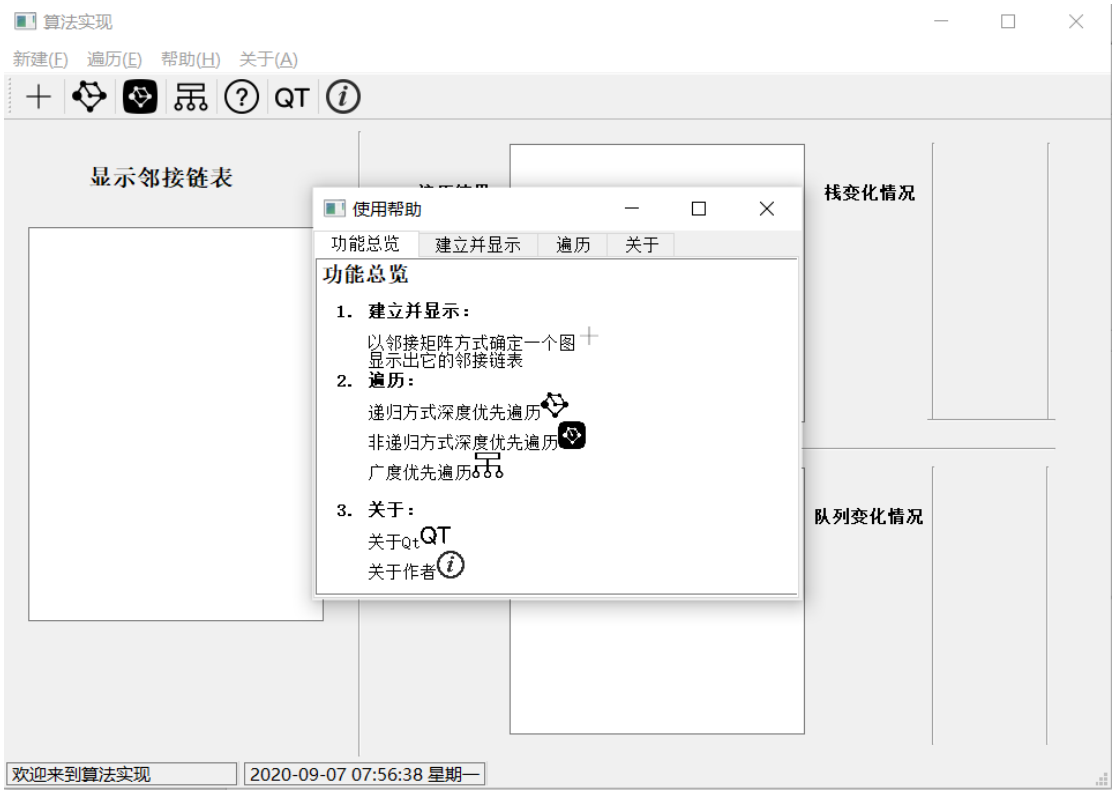
点击相应的按钮进入相应的功能，鼠标悬停在按钮上有相关提示，底部状态栏也有部分

提示。



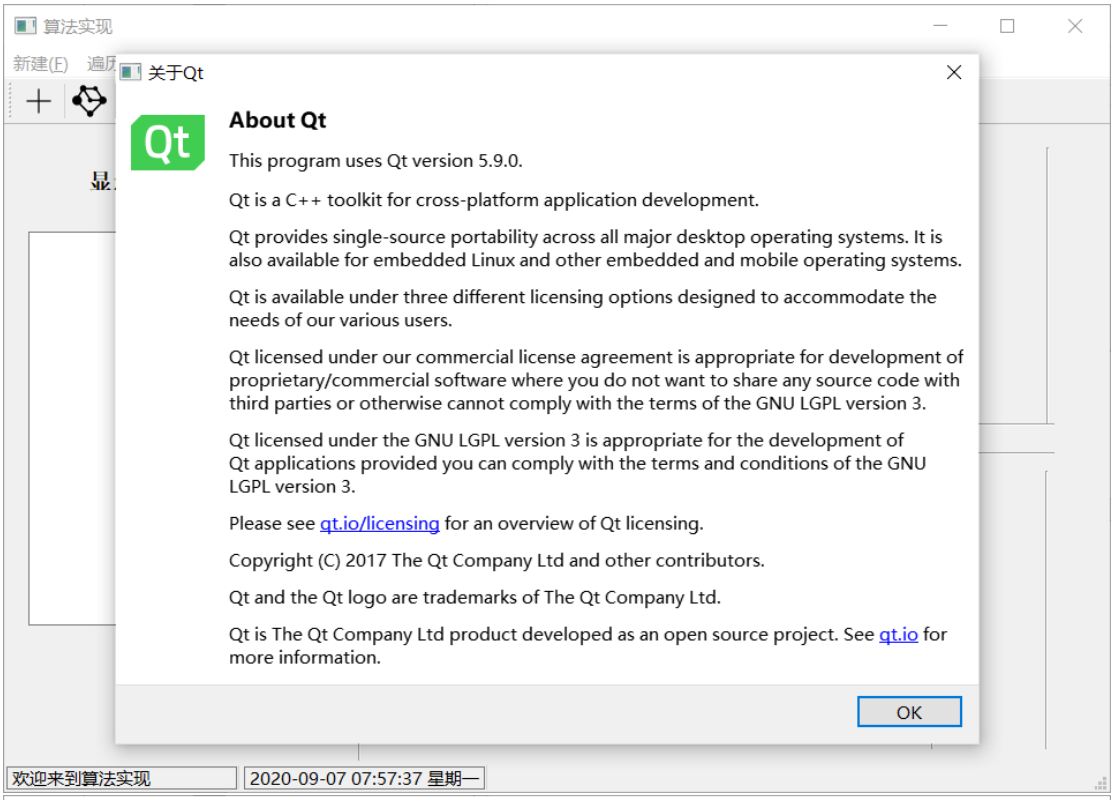
(3)查看帮助

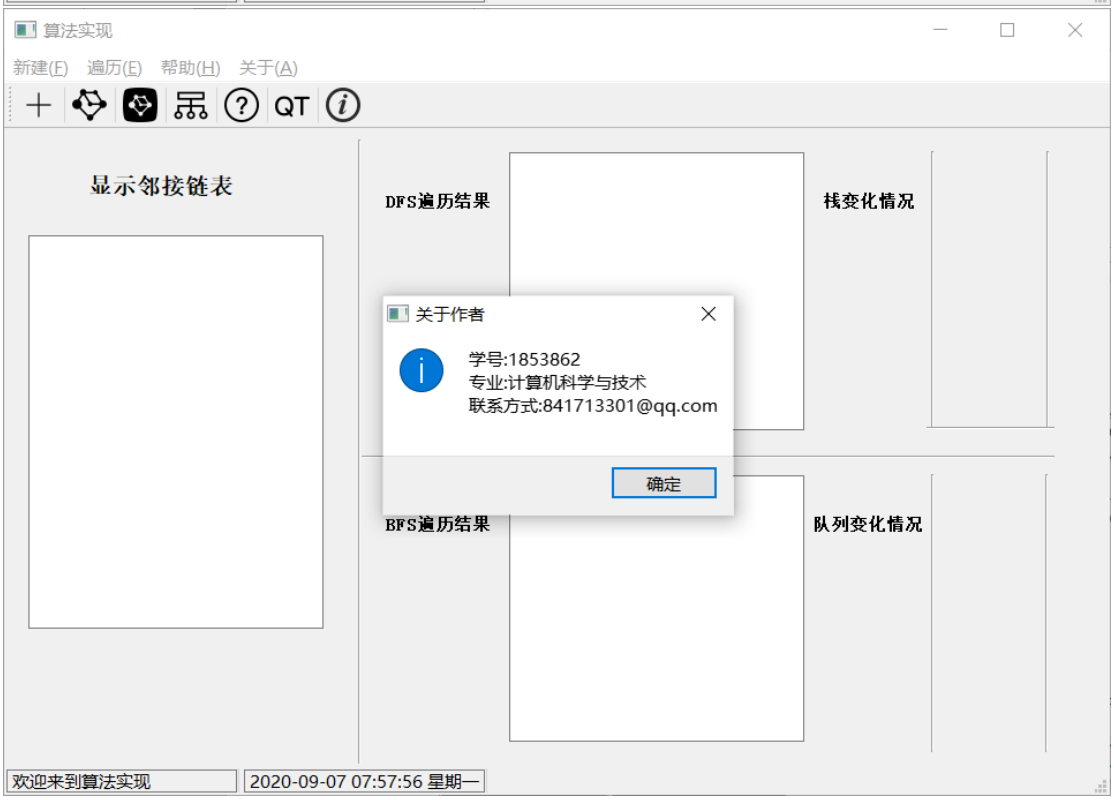
点击使用帮助按钮，查看使用帮助。



(4)查看关于

点击关于 Qt 和关于作者可查看有关信息。



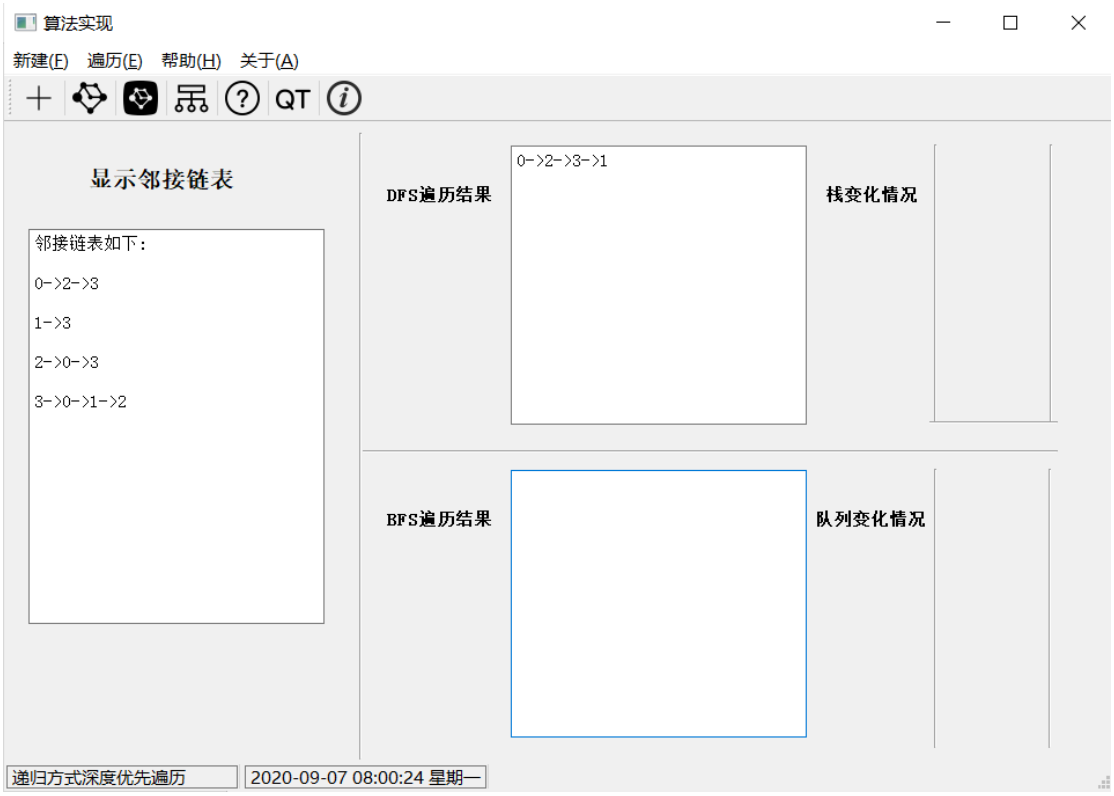


(5)建立新图

点击建图按钮可以以邻接矩阵方式确定一个图。

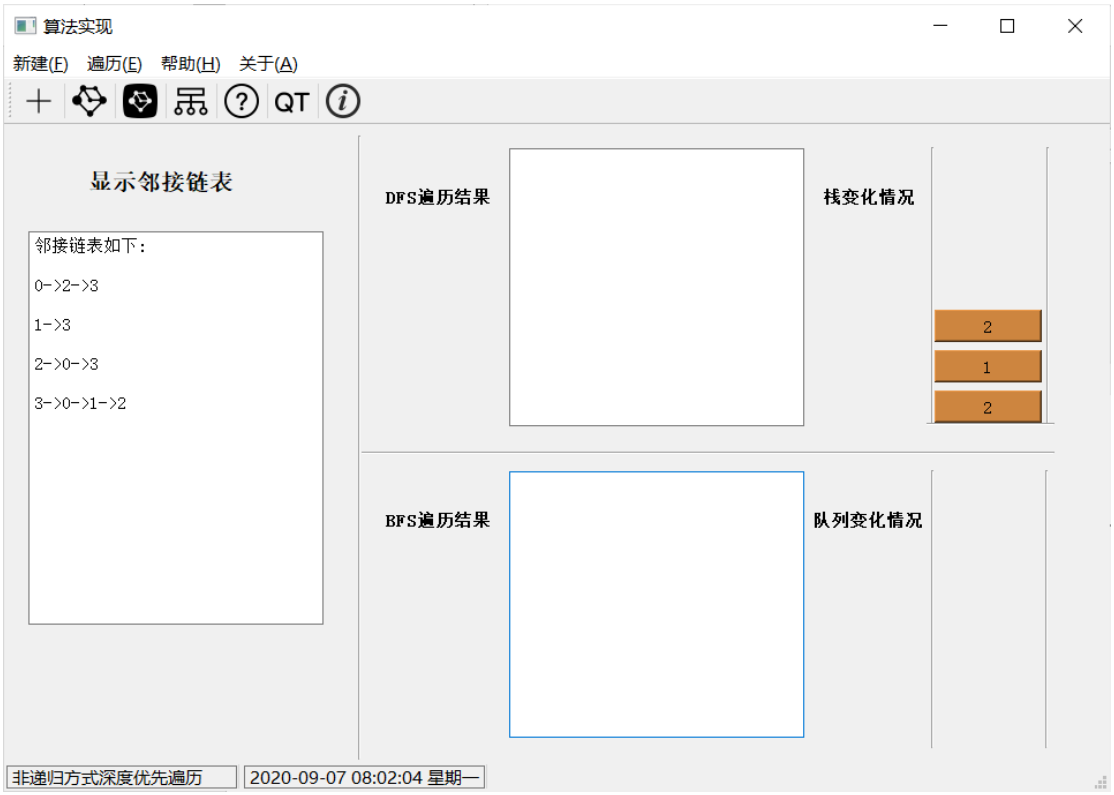
(6)递归方式深度优先遍历

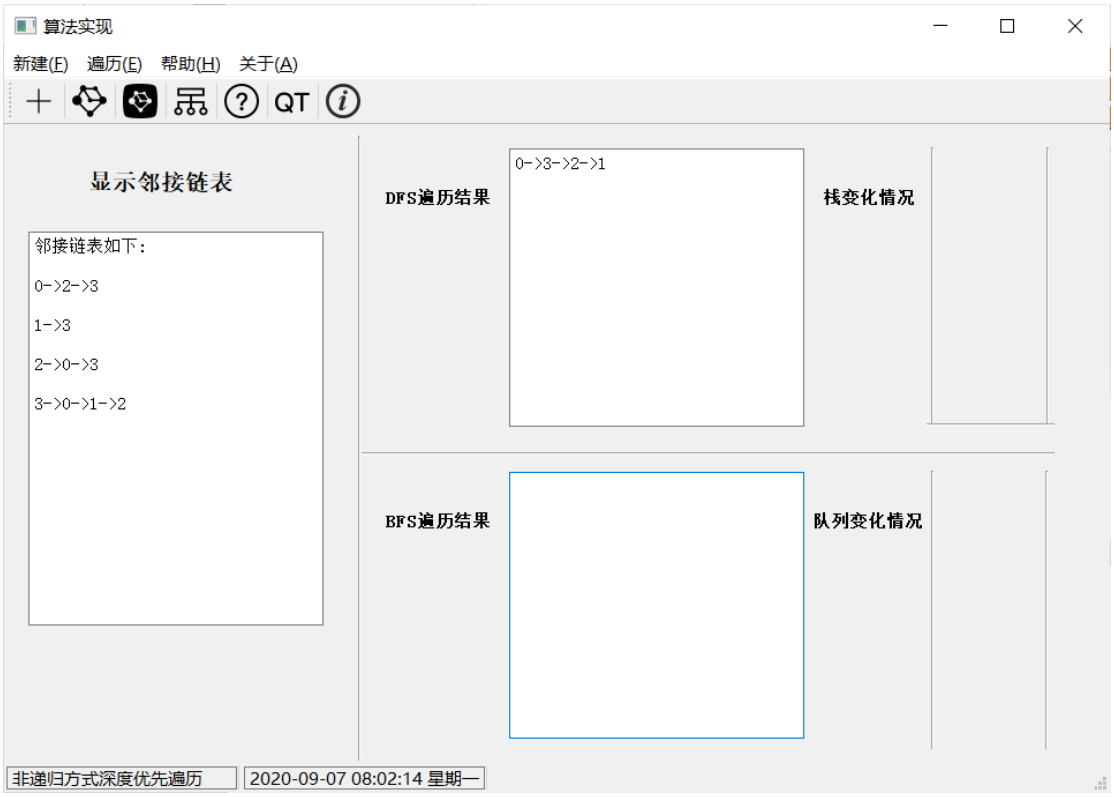
点击递归深度优先遍历按钮可进行深度优先遍历。



(7)非递归方式深度优先遍历

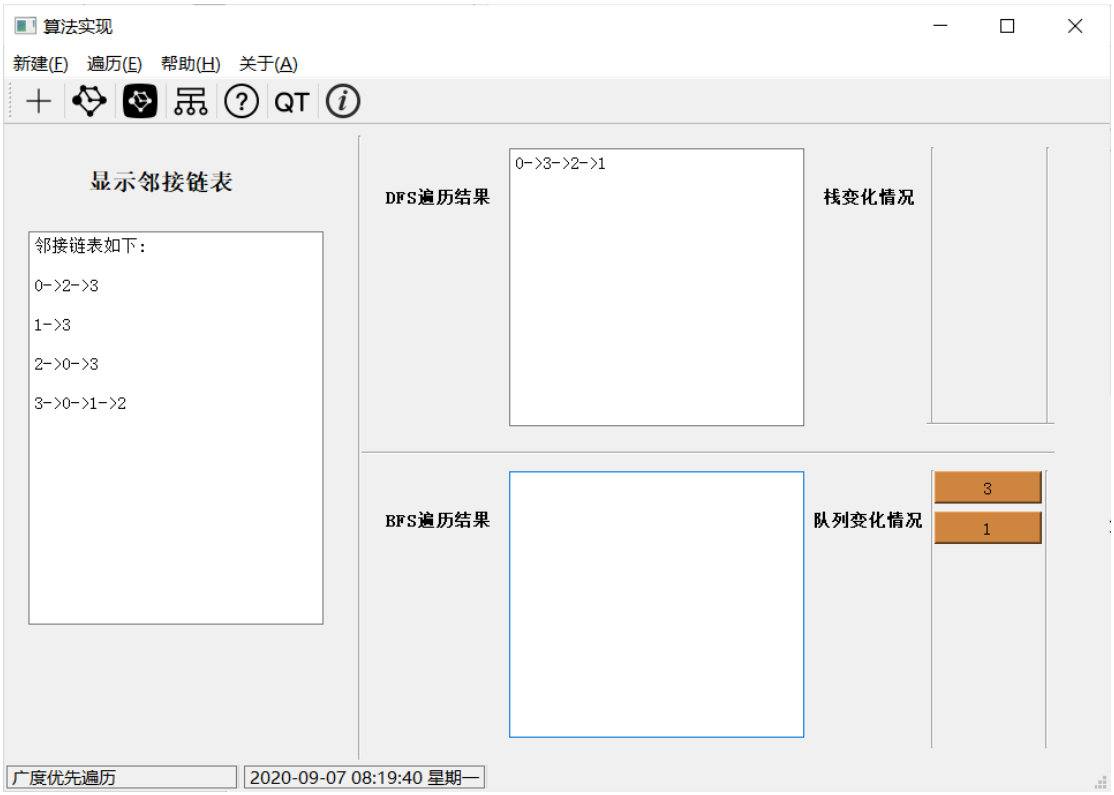
点击非递归深度优先遍历按钮可进行非递归深度优先遍历，先出现栈动态变化，栈变化结束后出现结果。

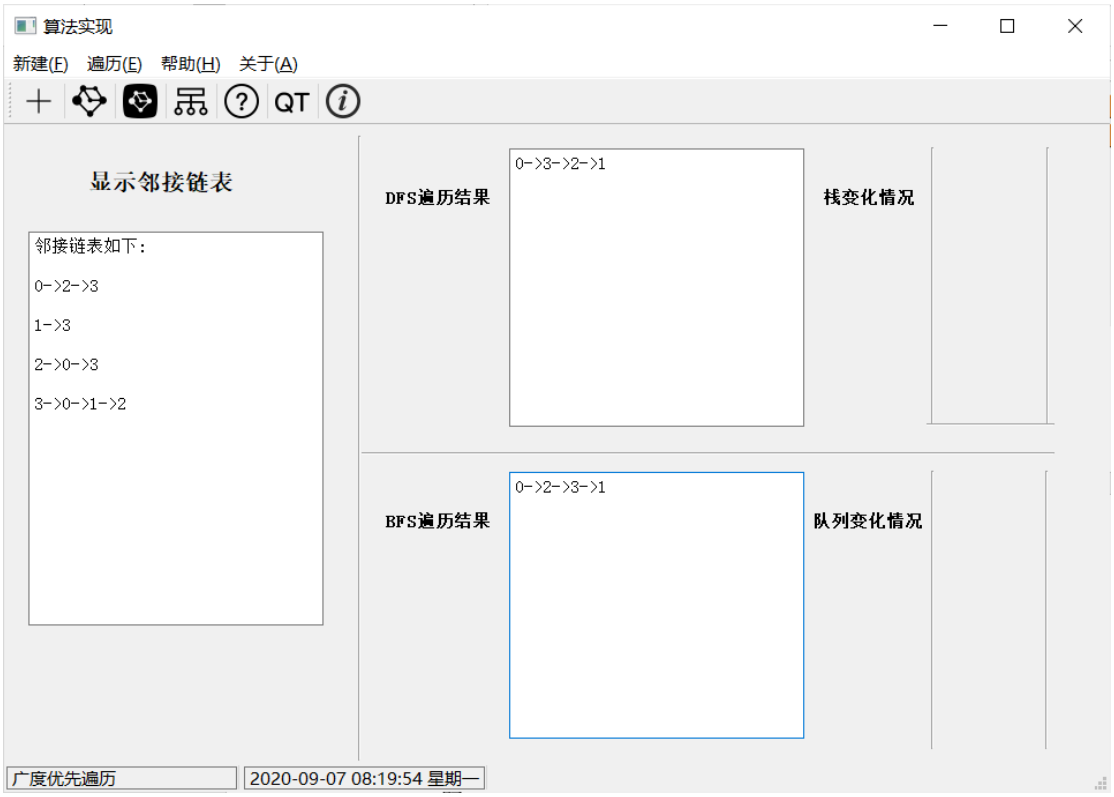




(8)广度优先遍历

点击广度优先遍历按钮可进行广度优先遍历，先出现队列动态变化，队列变化结束后出现结果。





第二部分 综合应用设计说明

2.1 题目

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

由于在每个功能实现方式部分都存在大量重复的后端接口调用，因此将通过文本方式说明，不再展示过多的后端代码；同时关于信号和槽的连接在此一并给出，之后不再赘述。

```
//连接信号和槽
void MainWindow::signal_slot()
{
```

```
    connect(dynamicadd->ui->tabWidget, &QTabWidget::currentChanged, this,
&MainWindow::tabWidgetupdate);
    connect(dynamicadd->ui->pushButtonline, &QPushButton::clicked, this,
&MainWindow::addline);
    connect(dynamicadd->ui->pushButtonstation, &QPushButton::clicked, this,
&MainWindow::addstation);
    connect(dynamicadd->ui->pushButtonconnect, &QPushButton::clicked, this,
&MainWindow::addconnect);
    connect(dynamicadd->ui->pushButtontext, &QPushButton::clicked, this,
&MainWindow::addtext);
    connect(ui->comboBoxstartline, static_cast<void(QComboBox::*)(const QString
&)>(&QComboBox::currentIndexChanged), this,
&MainWindow::transfer_startstation_update);
    connect(ui->comboBoxendline, static_cast<void(QComboBox::*)(const QString
&)>(&QComboBox::currentIndexChanged), this,
&MainWindow::transfer_endstation_update);
    connect(ui->pushButtonTransfer, &QPushButton::clicked, this,
&MainWindow::transferinquiry);
    connect(timer, &QTimer::timeout, this, &MainWindow::timerupdate);
}
```

(1)通过点击按钮、鼠标滚轮、键盘快捷键进行放大缩小

1）点击相应按钮调用下面的槽函数进行放大缩小

```
//视图放大
void MainWindow::on_actionenlarge_triggered()
{
    statuslabel1->setText(tr("视图放大"));
    ui->graphicsView->scale(1.25, 1.25);
}

//视图缩小
void MainWindow::on_actionnarrow_triggered()
{
    statuslabel1->setText(tr("视图缩小"));
    ui->graphicsView->scale(0.8, 0.8);
}
```

2）滚轮放大缩小通过事件过滤器实现；

```
bool Mouseaction::eventFilter(QObject *object, QEvent *event)
{
```

```
    //鼠标移动获取 viewpos 和 scenepos
    if (event->type() == QEvent::MouseMove)
    {
        QMouseEvent* mevent = static_cast<QMouseEvent*>(event);
        viewpos = mevent->pos();
        scenepos = myview->mapToScene(mevent->pos());
    }
    //滚轮移动完成缩放
    else if (event->type() == QEvent::Wheel)
    {
        QWheelEvent* wevent = static_cast<QWheelEvent*>(event);
        if (QApplication::keyboardModifiers() == Qt::NoModifier &&
wevent->orientation() == Qt::Vertical)
        {
            double scalex = qPow(1.0015, wevent->angleDelta().y());
            myview->scale(scalex, scalex);
            QPointF delta = myview->mapFromScene(scenepos) - viewpos;
            QPointF viewcenter = delta + QPointF(myview->viewport()->width() /
2.0,
                                                 myview->viewport()->height() /
2.0);;
            myview->centerOn(myview->mapToScene(viewcenter.toPoint()));
            return true;
        }
    }
    //表明未使用，消除警告
    Q_UNUSED(object)
    return false;
}
```

(2)查看地铁线路总图、查看线路信息、站点信息

1）点击相应按钮调用下面的槽函数查看路线总图；

```
//地铁线路总图
void MainWindow::on_actionmetronet_triggered()
{
    statuslabel1->setText(tr("上海地铁线路图"));
    scene->clear();
    QList<int> stationsList;
    QList<Edge> edgesList;
    manager->getgeneralroadmap(stationsList,edgesList);
    drawline(edgesList);
    drawstation(stationsList);
```

```
}
```

2）查看路线信息和站点信息在绘制图的点和边时实现；

```
QString tooltip=
"途经:
"+manager->getstationname(edgelist[i].first)+"-"+manager->getstationn
ame(edgelist[i].second)+
"\n""线路: "+getlinenames(linesList);

QString tooltip=
"站名: "+name+"\n"+
"经度: "+QString::number(longiLati.x(),'f',7)+"\n"+
"纬度: "+QString::number(longiLati.y(),'f',7)+"\n"+
"线路: "+getlinenames(linesList);
```

(3)动态添加线路、站点、连接，文本方式添加

1）点击相应按钮进入动态添加；

```
//动态添加
void MainWindow::on_actionadd_triggered()
{
    statuslabel1->setText(tr("动态添加"));
    dynamicadd->setdefault();
    dynamicadd->show();
}
```

2）在 dynamicadd 部分获取添加的数据，通过按钮传回 Mainwindow，然后通过 addline、addstation、addconnect、addtext 等槽函数完成添加；下面只展示 addline 槽函数，其他槽函数同理；

```
//添加线路功能函数
void MainWindow::addline()
{
    QMessageBox box;
    box.setWindowTitle(tr("添加线路"));
    if(dynamicadd->linename.isEmpty())
    {
        box.setIcon(QMessageBox::Warning);
        box.setText(tr("请输入线路名称"));
    }
    else if(manager->getlinehash(dynamicadd->linename)!=-1)
```

```
    {
        box.setIcon(QMessageBox::Warning);
        box.setText(tr("线路名已存在"));
    }
    else
    {
        box.setIcon(QMessageBox::Information);
        box.setText(tr("线路：")+dynamicadd->linename+tr("添加成功"));
        manager->addline(dynamicadd->linename, dynamicadd->linecolor);
        transfer_line_update();
    }
    box.addButton(tr("确定"),QMessageBox::AcceptRole);
    if(box.exec()==QMessageBox::Accepted)
    {
        box.close();
    }
}
```

(4)最短时间换乘策略、最少换乘次数换乘策略

1）点击相应按钮进行换乘，过程中通过 getmintimetransfer、getmintransferline 获取后端换乘数据，并将换乘策略绘制；

```
//换乘查询
void MainWindow::transferinquiry()
{
    int s1=manager->getstationhash(ui->comboBoxstartstation->currentText());
    int s2=manager->getstationhash(ui->comboBoxendstation->currentText());
    bool way=ui->radioButtonmintime->isChecked();
    if(s1==-1||s2==-1)
    {
        QMessageBox box;
        box.setWindowTitle(tr("查询错误"));
        box.setIcon(QMessageBox::Warning);
        box.setText(tr("线路没有"));
        box.addButton(tr("确定"),QMessageBox::AcceptRole);
        if(box.exec()==QMessageBox::Accepted)
        {
            box.close();
        }
    }
    else
    {
```

```
        QList<int> stationlist;
        QList<Edge> linelist;
        bool flag;
        if(true==way)
        {
            flag=manager->getmintimetransfer(s1, s2, stationlist, linelist);
        }
        else
        {
            flag=manager->getmintransferline(s1, s2, stationlist, linelist);
        }
        if(flag)
        {
            statuslabel1->setText(tr("换乘查询成功"));
            scene->clear();
            drawline(linelist);
            drawstation(stationlist);
            QString text=way?("以下线路时间最短\n\n"):("以下线路换乘最少\n\n");
            for(int i=0; i<stationlist.size(); ++i)
            {
                text+=manager->getstationname(stationlist[i]);
                if(i!=stationlist.size()-1)
                text+="\n ↓\n";
            }
            ui->textBrowser->clear();
            ui->textBrowser->setText(text);
        }
        else
        {
            QMessageBox box;
            box.setWindowTitle(tr("查询错误"));
            box.setIcon(QMessageBox::Warning);
            box.setText(tr("换乘无法到达终点站"));
            box.addButton(tr("确定"),QMessageBox::AcceptRole);
            if(box.exec()==QMessageBox::Accepted)
            {
                box.close();
            }
        }
    }
}
```

(5)查看使用帮助

1）点击按钮调用下面的槽函数；

```
//使用帮助
void MainWindow::on_actionhelp_triggered()
{
    usehelp->show();
}
```

(6)查看关于 Qt 和关于作者

1）点击按钮调用下面的槽函数；

```
//关于 Qt
void MainWindow::on_actionaboutqt_triggered()
{
    QMessageBox::aboutQt(this,tr("关于 Qt"));
}

//关于作者
void MainWindow::on_actionaboutauthor_triggered()
{
    QMessageBox box;
    box.setWindowTitle(tr("关于作者"));
    box.setIcon(QMessageBox::Information);
    box.setText(tr("学号:1853862\n"
                   "专业:计算机科学与技术\n"
                   "联系方式:841713301@qq.com\n"));
    box.addButton(tr("确定"),QMessageBox::AcceptRole);
    if(box.exec() == QMessageBox::Accepted)
        box.close();
}
```

2.3 设计思想

(1)实现思路

1）分析题目要求，进行顶层设计

首先对题目要求进行分析，看所需实现的功能有哪些，所需要的页面有哪些，页面如何布局，数据结构如何选择等，有一个较好的整体设计，然后再考虑细化的问题。

2）前后端分离

软件开发过程采用前后端分离。将数据的处理、逻辑的实现等放到后端；将页面的显示，用户的交互等放到前端；前后端通过接口连接或者前端主导后端数据的调用，形成整体的架构。

3）后端实现

软件开发从后端入手，利用之前熟悉的 VS、VSCode 等控制台应用开发工具，将后端实现，在控制台查看数据结构是否合适、算法设计是否合理、最终结果是否符合预期。

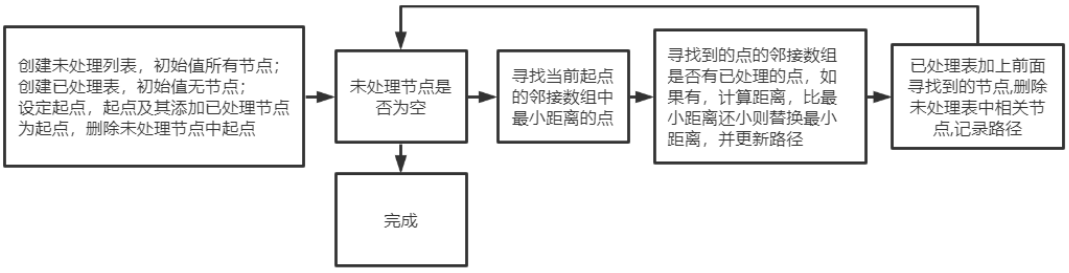
4）前端分模块实现

在后端设计完成后，对于前端的每个页面进行分别的 UI 设计，然后逐步实现每一个功能，将前后端对接起来，完成整个程序。

(2)算法设计基本流程

本软件除去部分前后端接口外，每个后端处理函数都可看作一个算法，其中有大量的简单算法。对于简单算法，每个算法之间耦合性不强，只需要注意一下每个算法的前后顺序和数据传递。这些简单算法依次设计，依次调试即可。

有一个核心算法为迪杰斯特拉算法，该算法流程图如下：



2.4 逻辑结构与物理结构

(1)逻辑结构

本软件主要采用的逻辑结构是集合结构、线性结构和图形结构；

```
typedef QPair<int,int> Edge;//定义边类型

//线路类
struct Line
{
    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> stationset;    //线路站点集合
    QSet<Edge> edges;        //线路站点连接关系
//地铁站点类定义
struct Station
{
    int id;                  //站点 ID
    QString name;            //站点名称
    double longitude, latitude;//站点经纬度
    QSet<int> theline;       //站点所属线路
//图的邻接点结构
struct Node
{
    int station;    //邻接点 ID
    double distance;//两点距离
//后端
class Manager
{
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
    double theminLongitude, theminLatitude, themaxLongitude, themaxLatitude;//
最小最大经纬度
```

(2)物理结构

本软件主要采用的物理结构有顺序结构、链式结构和散列结构，如上面代码所示。

2.5 开发平台

(1)开发平台

机器型号：联想 Y7000(2018 款)

CPU：Intel Core i5 8300H

内存：8GB

操作系统：Windows 10

开发语言：C++

编译器：MinGW 32bit

开发框架：QT 5.9.0

IDE：Qt Creator 4.3.0

(2)运行环境

所提交的源代码可在上述 IDE 运行。

所提交的 EXE 文件可在 Windows 机型下运行。

2.6 系统的运行结果分析说明

(1)调试及开发过程

1）调试过程：

本软件主要采用 Qt Creator 进行开发调试，由于本软件涉及到大量图形界面和用户交互，所以断点调试、单步运行等方式并不适合。同时因为本软件的复杂逻辑较少，所以在调试过程中主要使用 qDebug 进行控制台输出完成调试。例如：

```
106  //获得线路表的名字集
107  QString MainWindow::getlinenames(const QList<int>& linelist)
108  {
109      QString str;
110      for (int i=0; i<linelist.size(); ++i)
111      {
112          str+=manager->getlinename(linelist[i]);
113          str+=" ";
114      }
115      qDebug()<<str;
116      return str;
117  }
118
```

```
应用程序输出
algorithm   application   SubwayTransferSystem2
Starting C:\Users\84171\git\SubwayTransferSystem\build-application-Des
"10号线 "
"3号线 "
"12号线 "
"4号线 "
"7号线 "
"12号线 "
"5号线 "
```

在控制台输出的是 linename，可以看到每条地铁线名称。

2）开发过程：

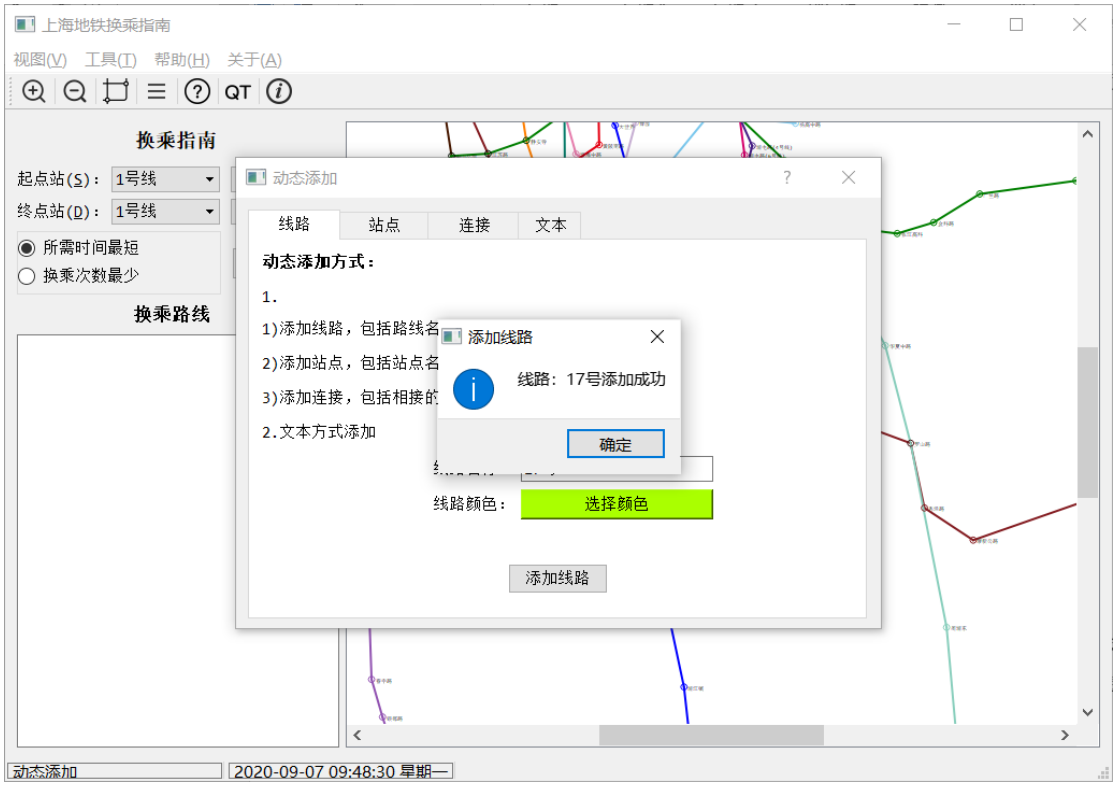
本软件先开发后端，在完成后端后再开发前端。前端根据页面设计，分别开发主页面、动态添加页面、使用帮助页面。然后将前后端进行对接，并对每个模块进行开发测试，调整错误，细化代码逻辑，最终完成全部开发。

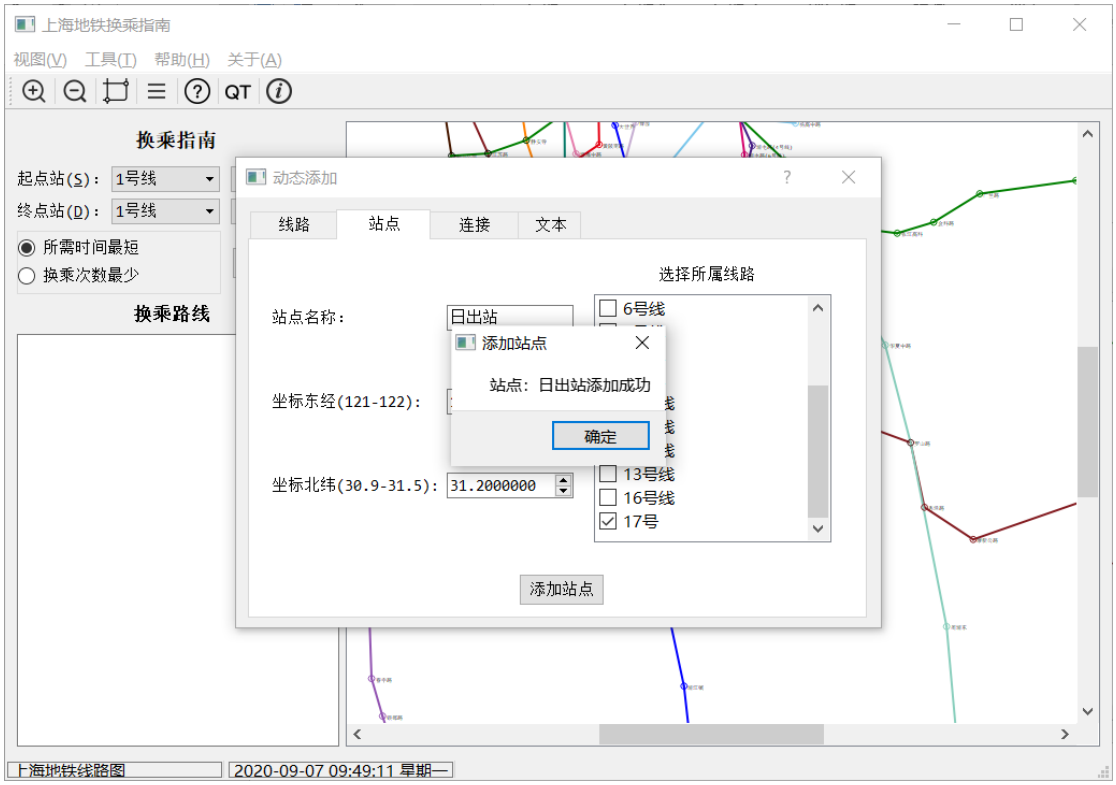
(2)开发成果

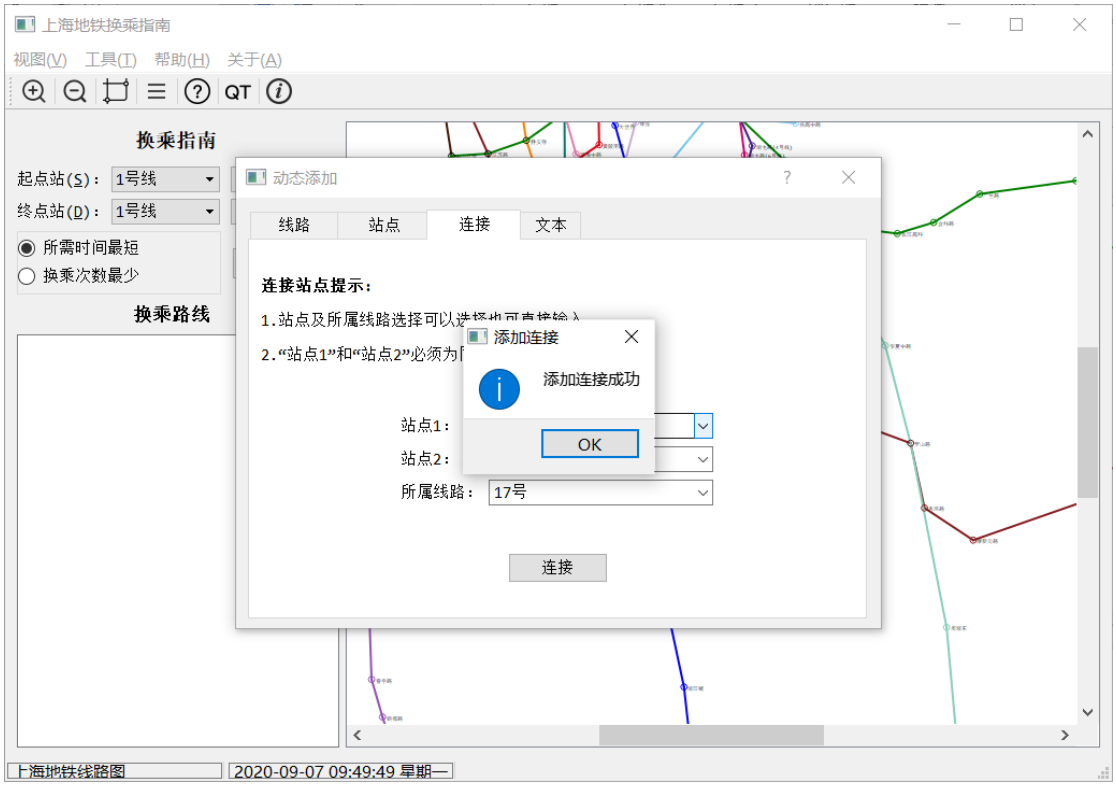
1）正确性：

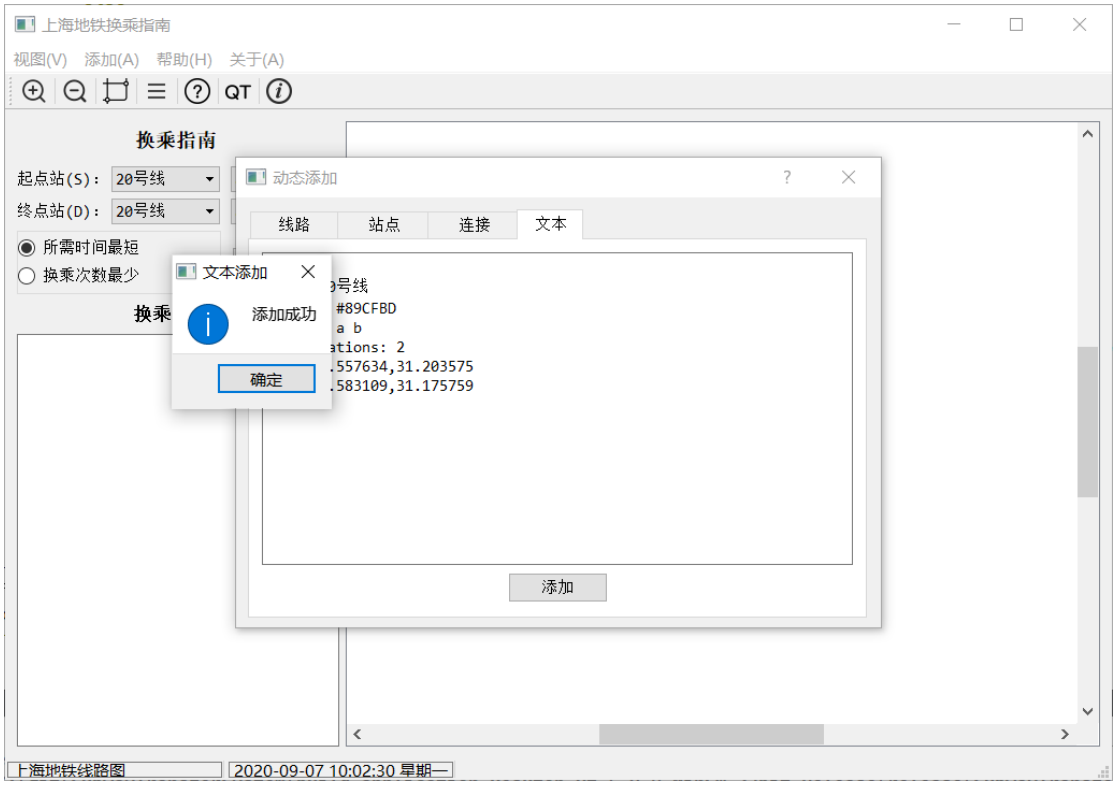
该软件的每个功能都保证了正确性。

动态添加：

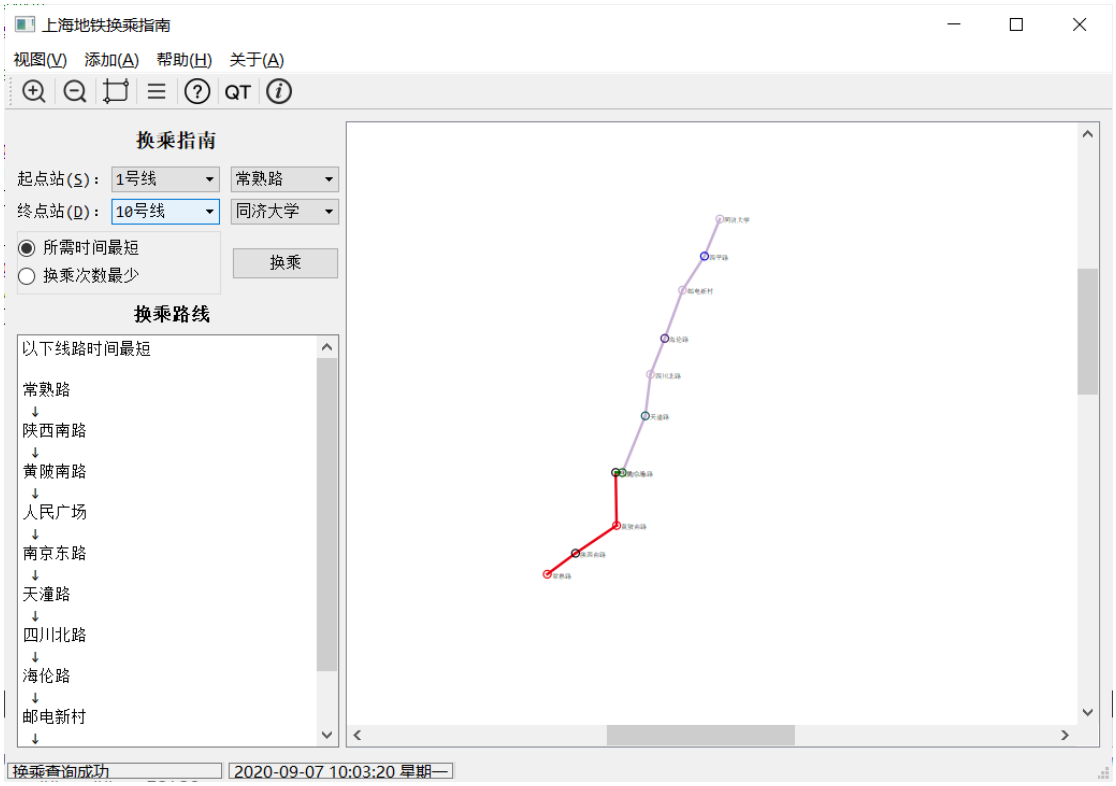


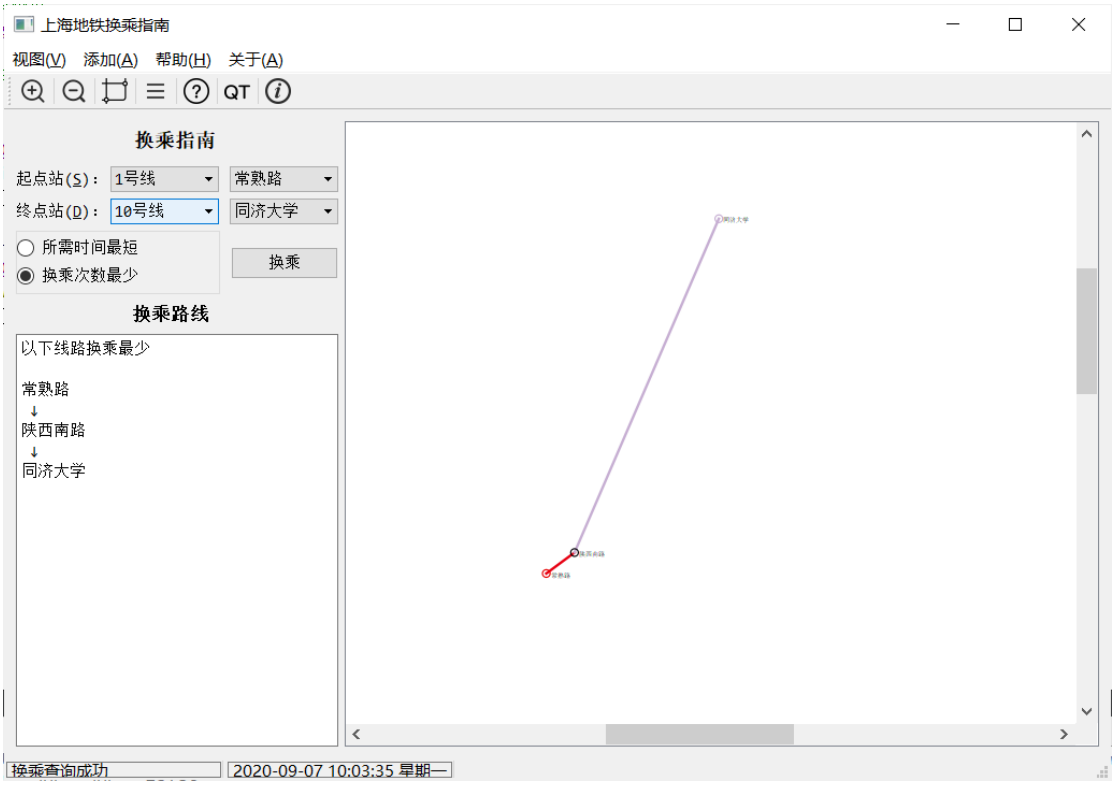

上海地铁换乘指南
视图(V) 工具(T) 帮助(H) 关于(A)
QT
换乘指南
起点站(S)： 1号线
终点站(D)： 1号线
所需时间最短
换乘次数最少
换乘路线
动态添加
线路 站点 连接 文本
选择所属线路
站点名称： 日出站
添加站点
站点：日出站添加成功
确定
坐标东经(121-122)：
坐标北纬(30.9-31.5)： 31.2000000
6号线
13号线
16号线
17号
添加站点
上海地铁线路图
2020-09-07 09:49:11 星期一


上海地铁换乘指南
视图(V) 工具(T) 帮助(H) 关于(A)
QT
换乘指南
起点站(S)： 1号线
终点站(D)： 1号线
所需时间最短
换乘次数最少
换乘路线
动态添加
线路 站点 连接 文本
连接站点提示：
1.站点及所属线路选择可以
2.“站点1”和“站点2”必须为
添加连接
添加连接成功
OK
站点1：
站点2：
所属线路： 17号
连接
上海地铁线路图
2020-09-07 09:49:49 星期一


上海地铁换乘指南
视图(V) 添加(A) 帮助(H) 关于(A)
QT
换乘指南
起点站(S)： 20号线
终点站(D)： 20号线
所需时间最短
换乘次数最少
换乘
动态添加
线路 站点 连接 文本
文本添加
添加成功
确定
号线
#89CFBD
a b
ations: 2
.557634,31.203575
.583109,31.175759
添加
上海地铁线路图
2020-09-07 10:02:30 星期一

换乘：


上海地铁换乘指南
视图(V) 添加(A) 帮助(H) 关于(A)
QT
换乘指南
起点站(S)： 1号线
常熟路
终点站(D)： 10号线
同济大学
所需时间最短
换乘次数最少
换乘
换乘路线
以下线路时间最短
常熟路
↓
陕西南路
↓
黄陂南路
↓
人民广场
↓
南京东路
↓
天潼路
↓
四川北路
↓
海伦路
↓
邮电新村
↓
换乘查询成功
2020-09-07 10:03:20 星期一



2）稳定性：

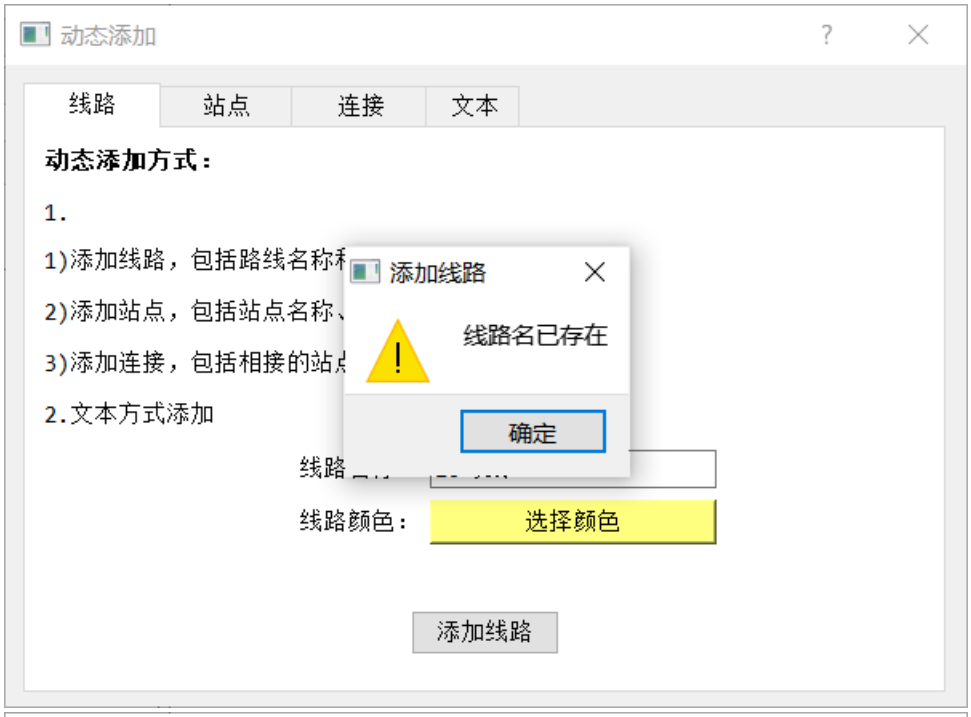
程序稳定性良好。一方面程序运行速度正常，所有操作都得到迅速响应，而且不会出现程序卡住未响应的情况；另一方面程序可以稳定多次的添加路线站点连接等；再一方面，添加后数据会正常更新。

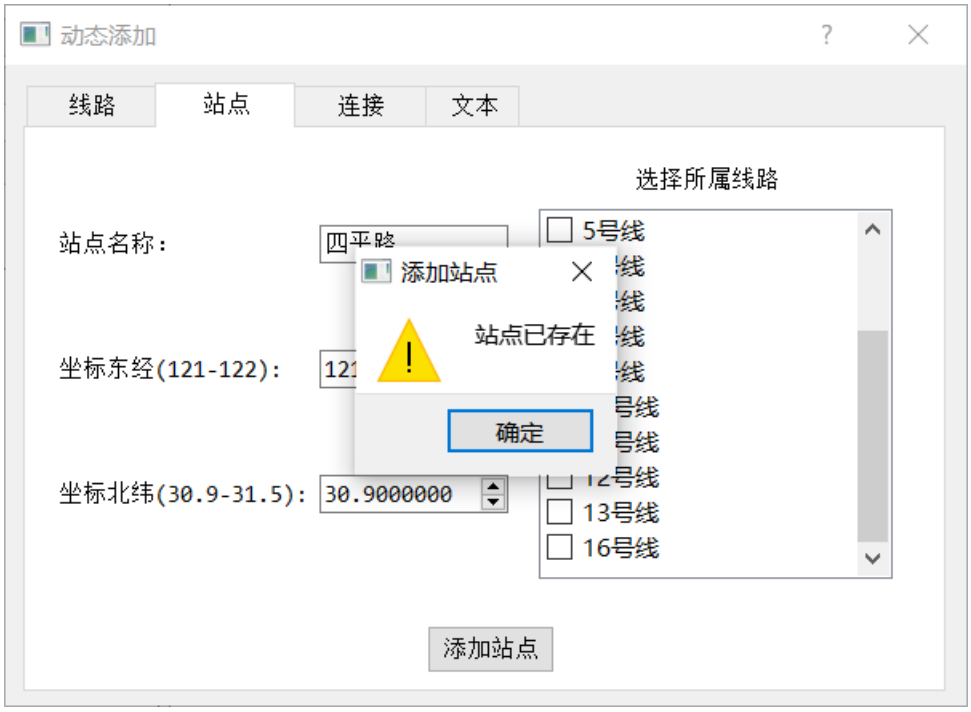
经过多次全分支覆盖的测试，程序未出现异常。

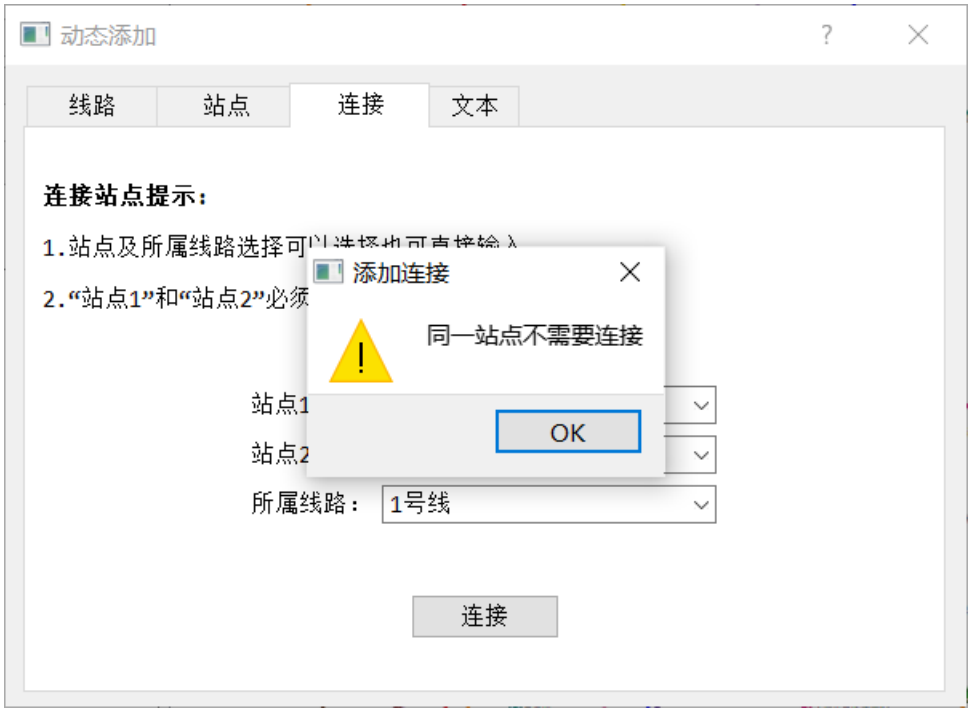
3）容错能力：

程序容错能力良好。

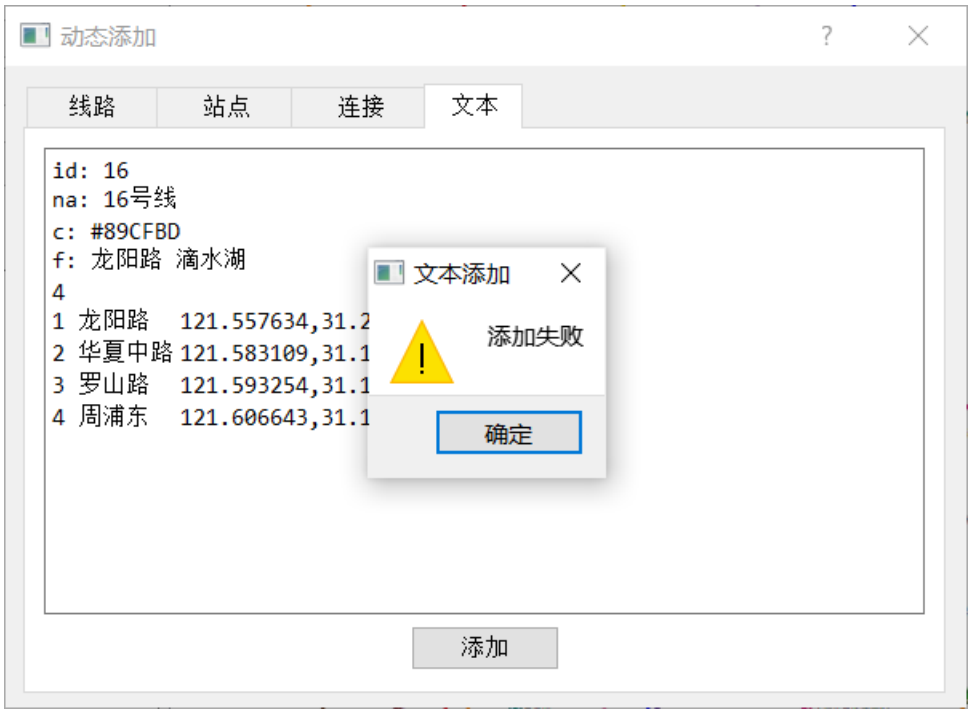
不能非法添加


动态添加
线路
站点
连接
文本
动态添加方式：
1.
1)添加线路，包括路线名称
2)添加站点，包括站点名称
3)添加连接，包括相接的站点
2.文本方式添加
添加线路
线路名已存在
确定
线路颜色：
选择颜色
添加线路

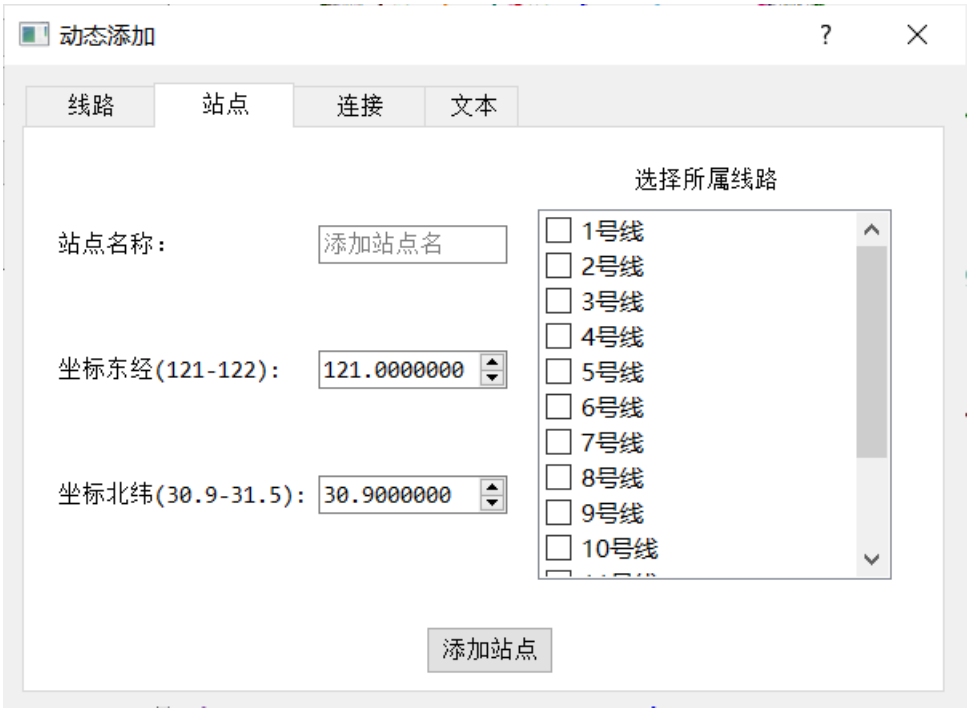

动态添加
线路
站点
连接
文本
选择所属线路
站点名称：
坐标东经(121-122)：
坐标北纬(30.9-31.5)：
30.9000000
添加站点
站点已存在
确定
5号线
13号线
16号线
添加站点



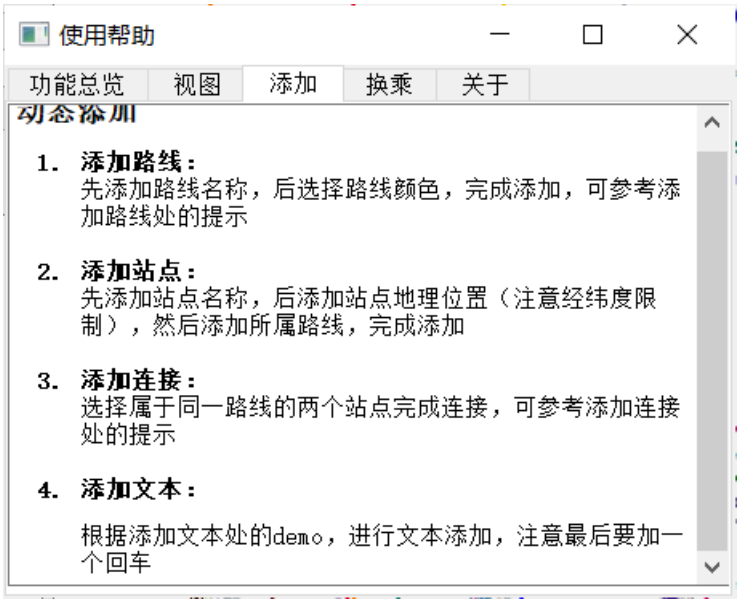
文本有格式限制；



数据添加有限制：



关于其他程序限制都写入了使用帮助；



(3)运行结果

运行案例请见下方操作说明。

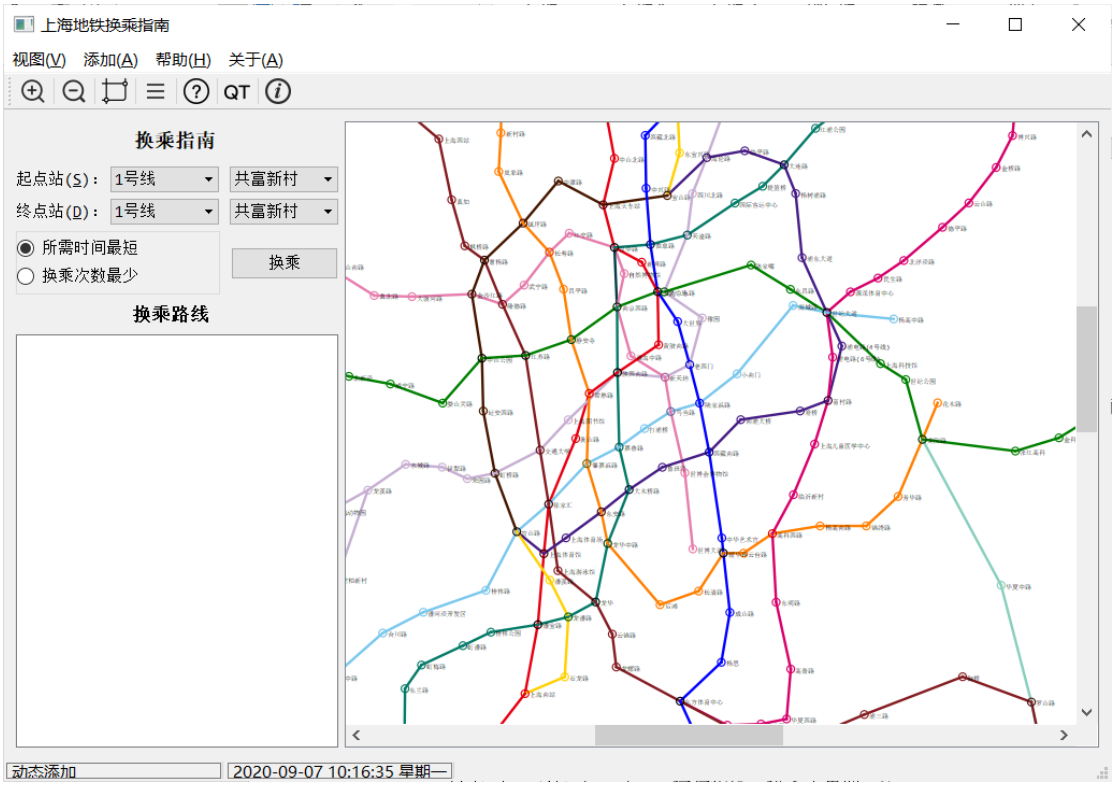
2.7 操作说明

(1)点击 EXE

点击 application_boxed.exe 运行程序，进入主界面。

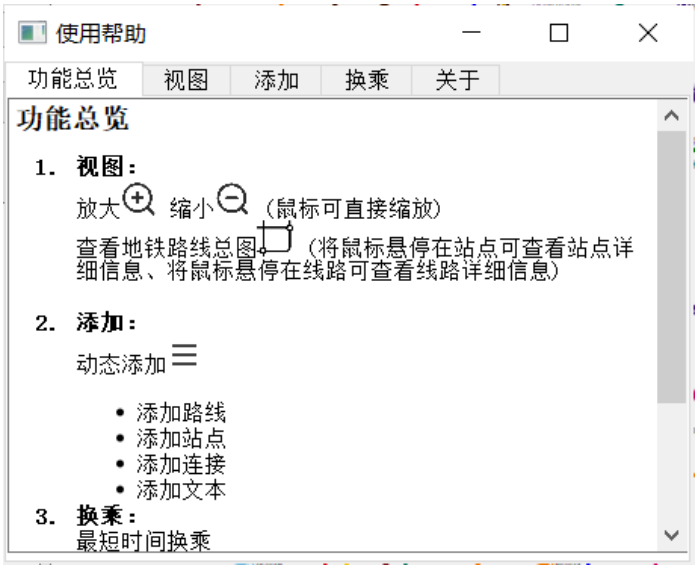
(2)主界面

点击相应的按钮进入相应的功能，鼠标悬停在按钮上有相关提示，底部状态栏也有部分提示。



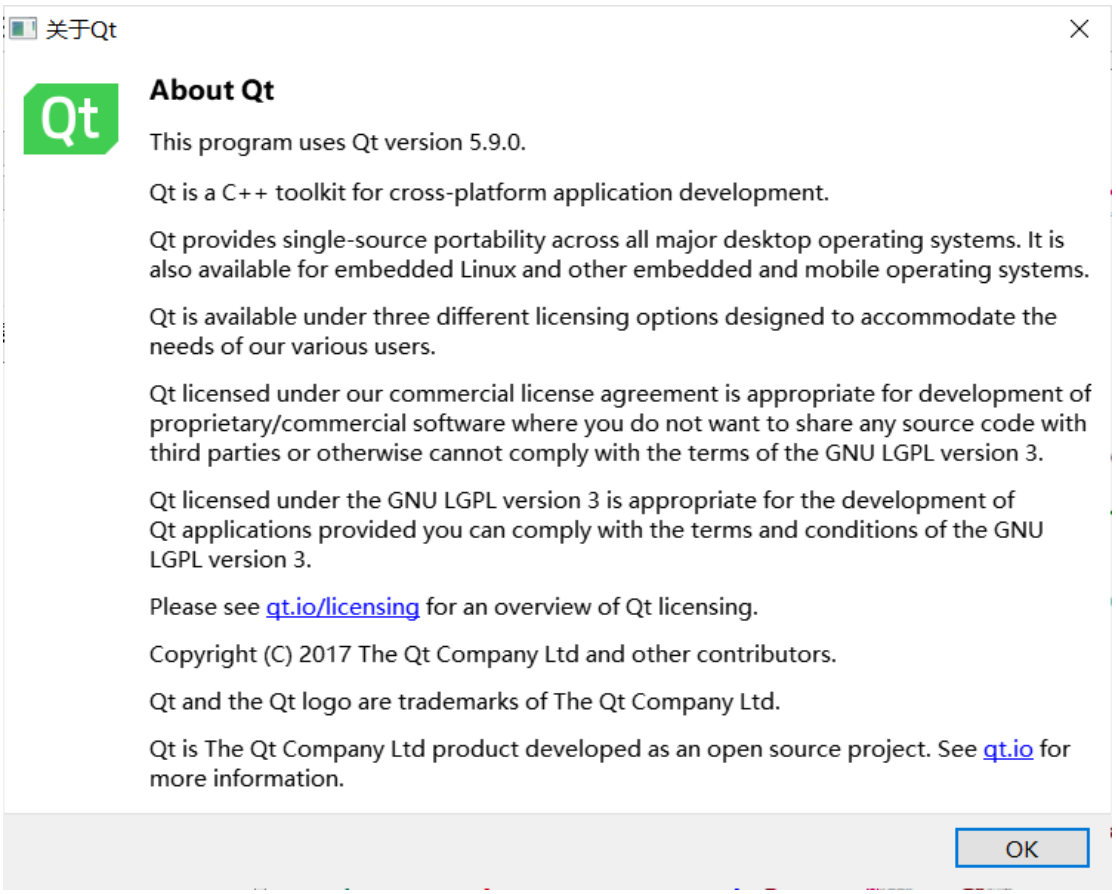
(3)查看帮助

点击使用帮助按钮，查看使用帮助。



(4)查看关于

点击关于 Qt 和关于作者可查看有关信息。

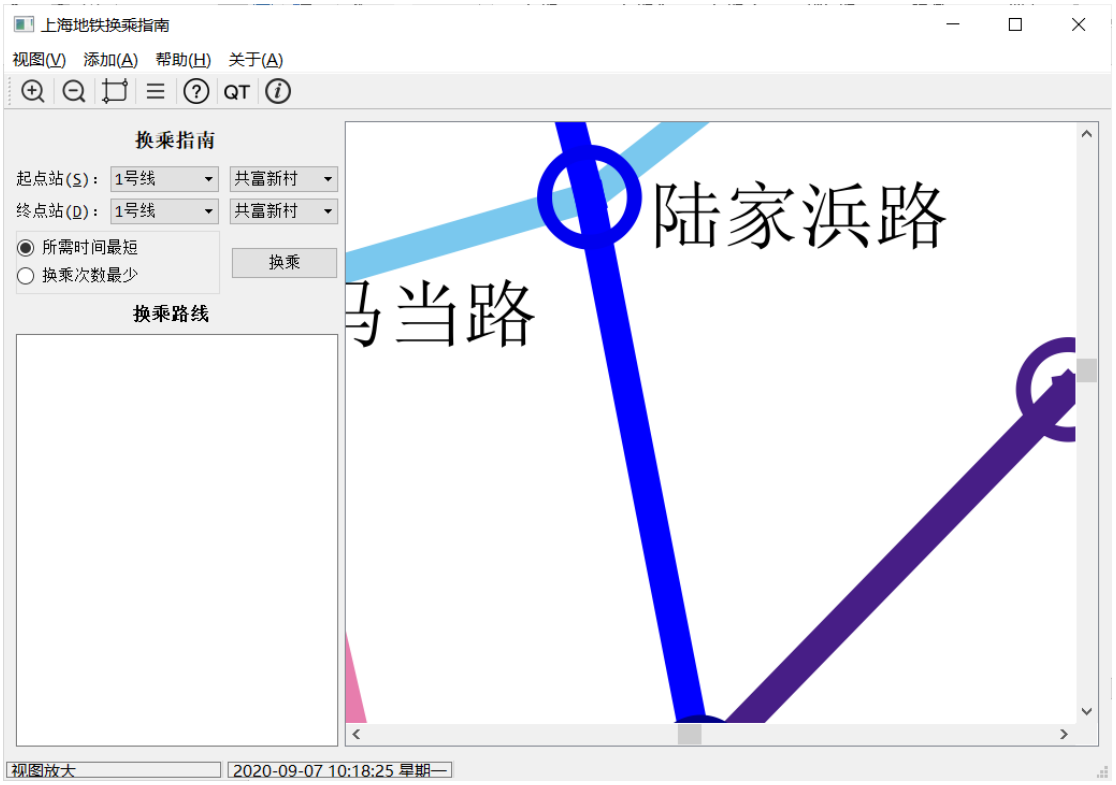


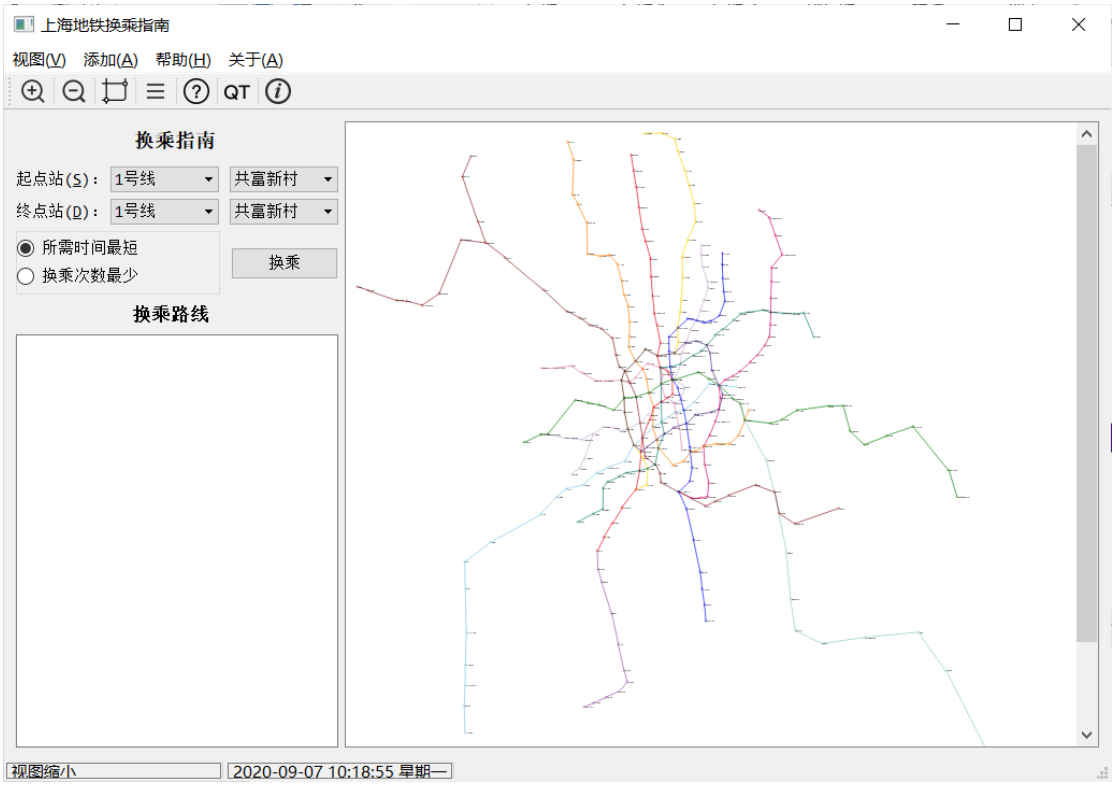
(5)视图放大缩小，查看地铁路线

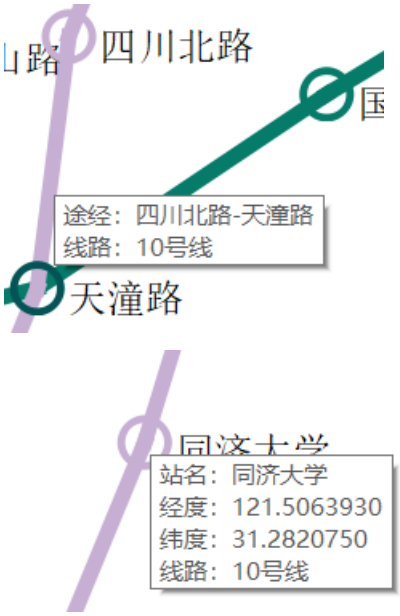
点击相应按钮或滚轮操作进行放大缩小；

点击相应按钮查看路线总图；

鼠标悬停查看路线、站点信息；

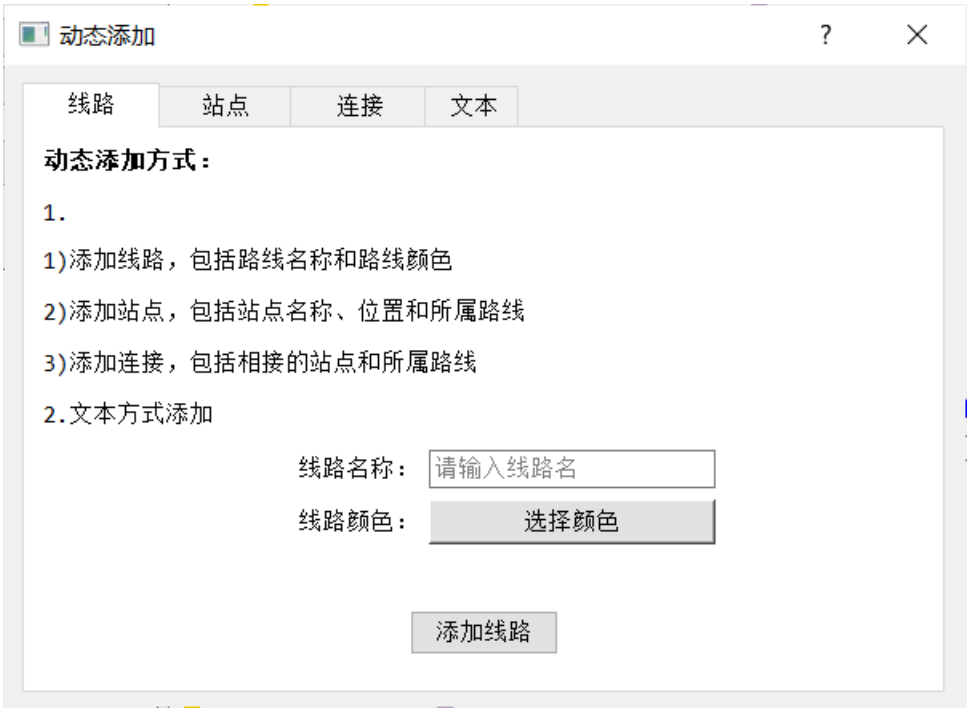

上海地铁换乘指南
视图(V) 添加(A) 帮助(H) 关于(A)
QT
换乘指南
起点站(S)： 1号线 共富新村
终点站(D)： 1号线 共富新村
所需时间最短
换乘次数最少
换乘
换乘路线
陆家浜路
马当路
视图放大
2020-09-07 10:18:25 星期一


上海地铁换乘指南
视图(V) 添加(A) 帮助(H) 关于(A)
QT
换乘指南
起点站(S)： 1号线 共富新村
终点站(D)： 1号线 共富新村
所需时间最短
换乘次数最少
换乘
换乘路线
视图缩小
2020-09-07 10:18:55 星期一



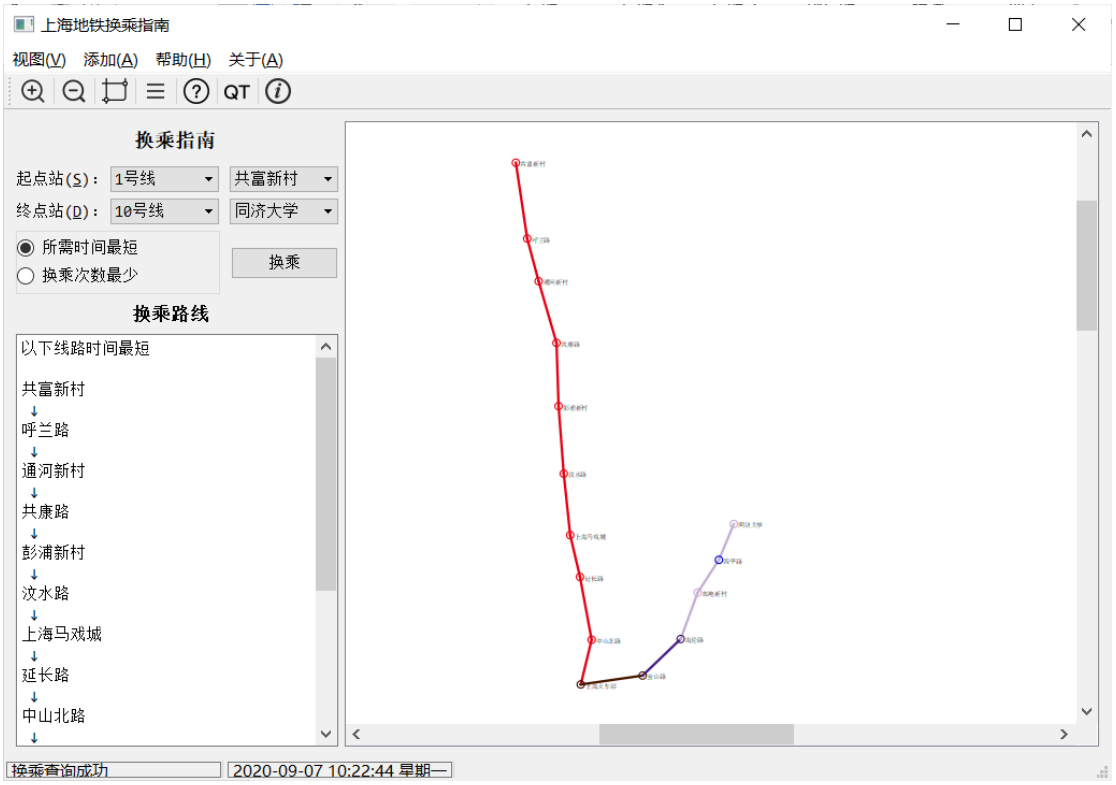
(6)动态添加

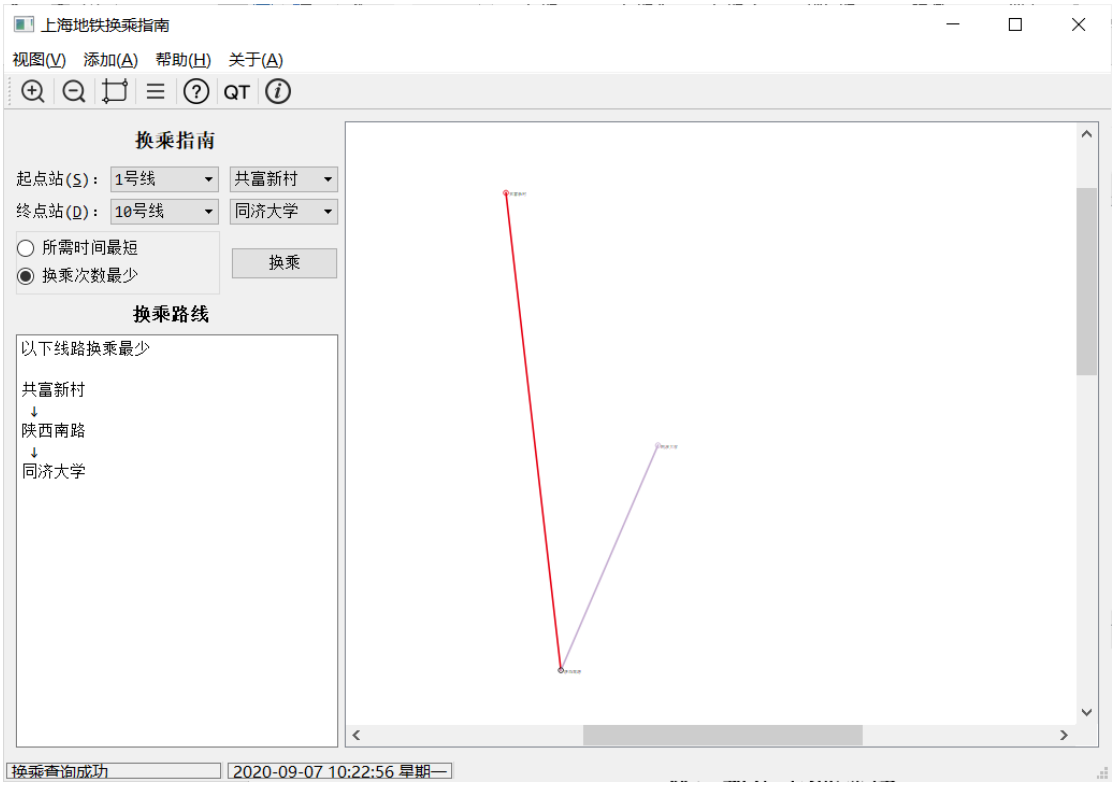
点击相应按钮进行动态添加，添加时根据页面提示即可，有疑问可查询使用帮助；



(7)换乘查询

选择起点、终点，点击换乘按钮查询换乘；





第三部分 实践总结

3.1.所做的工作

学习并运用 Qt5 相关知识；

复习、补充并运用数据结构相关知识；

学习并运用部分 C++11 新特性如 lamda 表达式等

独立完成算法实现软件设计；

独立完成综合应用软件设计；

完成课程设计报告一份；

3.2.总结与收获

(1)能力提升

一方面我认为自己的自学能力得到了很大的提高。这次课程设计是由自己来选择开发方法、开发框架等。我完成了自学 Qt5 的部分知识、自学 C++11 部分新特性、程序打包发行等，有效地提高了自学能力。

另一方面我认为自己地程序设计能力有了很大的提高。本次课程设计涉及到的内容较多，程序量大；在编程前期我进行了详细地设计，比如架构选择、页面布局、数据处理等，通过一系列设计使得最终的软件能平稳运行，有较好的容错能力，bug 较少。

(2)收获

课程设计从算法设计到综合应用，从简单系统到复杂系统，从之前熟悉的控制台应用程序到图形界面、用户交互，我对计算机的认识，对编程的了解得到了大幅提高。

通过前期学习 QT5 框架，信号和槽、基本控件、对象模型、布局、文件操作、事件、绘图、多线程等知识，知识储备得到了提高。

在课程设计过程中自我学习、自我设计、自我开发，自学能力、抗压能力、创新能力、调试能力等得到了提高。

同时，课程设计使得我从课本理论知识走向了实际应用，激发了我的学习和实践热情。在课程设计中发现问题、解决问题，激发了我继续求知，深度探索的想法，我希望能在之后的生涯里一直学习、一直进步，不断提高自己的技术水平。

(3)个人体会

1）我们必须先关注重点，先把核心功能实现，再逐步地修改细节，进行改进。

2）一定要注意编程细节，比如内存的释放时间，分支的覆盖，一般程序出现问题，往往都是在细节上有漏洞。

3）程序设计过程中要注重算法设计，要积极寻找各类不同的算法，进行实践比较。

4）一边学习一边实践，及时把所学的应用到实践中去，这样才能更有效率地学习，也能更有效率地实现。

5）搜索引擎是自学过程中的好老师。在遇到自己不了解的部分时，可以去cppreference、stackoverflow等地方寻找解决方法。

第四部分 参考文献

[1] 严蔚敏，吴伟民.数据结构（C语言版）[M].北京：清华大学出版社,2007

[2] StanleyB.Lippman, JoséeLajoie, BarbaraE.Moo. C++ Primer 中文版[M]. 电子工业出版社, 2013.

[3] 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

[4] 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

[5] 王晓东. 计算机算法设计与分析.第4版[M]. 电子工业出版社, 2012.