目录

1. 项目基本信息
2. 项目简介
3. 小组成员信息
4. 项目描述

2.1 问题描述

1. 拓扑排序应用的背景和重要性
2. 具体应用场景

2.2 系统功能性需求

1. 描述项目的主要功能
2. 说明图形界面的要求
3. 定义数据输入格式

2.3 系统非功能性需求

1. 指定系统的运行平台
2. 选择开发工具和开发语言
3. 项目涉及的底层技术说明

3.1UI界面

3.2图形生成

3.3图形保存

3.4数据导入

3.5帮助文档

3.6退出程序

3.7主函数

1. 可行性分析

4.1 技术可行性

4.2 时间可行性

1. 项目成果展示

5.1. 导入导出文件功能

5.2. 生成拓扑排序无权图功能

5.3. 生成拓扑排序有权图功能

5.4. 改变拓扑排序图的布局功能

5.5. 帮助文档的查看功能

5.6. 计算最短路径功能

5.7. 退出程序功能

1. 项目总结与思考

6.1. 软件设计的重要性

6.2. 数据处理与算法

6.3. 文件导入导出

6.4. 可视化

6.5. 帮助文档与用户支持

6.6. 扩展性

6.7. 实践和学习

1. 源代码
2. 项目基本信息
3. 项目简介

项目名称：拓扑排序交通路线规划应用

在城市交通系统中，公交线路的规划通常需要考虑站点之间的先后关系，以确能够最大程度地提高交通系统运转的效率。当今城市交通规划领域存在一个显著的痛点，即公交车站站点的复杂依赖关系往往以文字的形式呈现，而这对于交通规划者和城市规划者来说并不直观。这种情况加大了交通规划的难度，因为规划者需要手动分析和理解站点之间的复杂联系，以确定最佳的公交线路和路线规划。

图-1 深圳市公交线路图

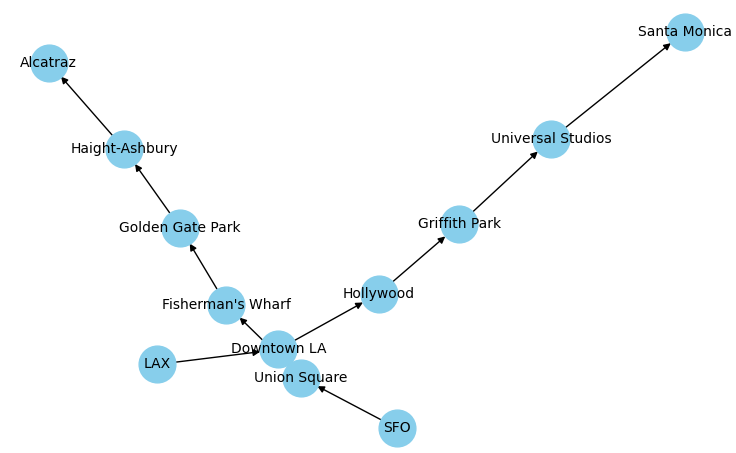
本项目旨在开发一个基于拓扑排序的交通和路线规划应用，以解决公交线路和站点之间的依赖关系，从而实现最佳路线的规划。该项目将为公共交通系统规划部门提供有力工具，以优化公交线路和提高乘客出行效率。

图-2 基于拓扑排序生成三条公交线路依赖关系图

项目的主要特点和功能包括：

* 用户友好的图形用户界面，使城市规划者和交通规划者能够轻松输入、可视化公交站点和保存分析结果与图像。
* 自动化拓扑排序算法，根据输入的站点数据计算出站点之间的依赖关系，并生成最佳的公交线路规划。
* 可视化结果展示，以图形方式呈现最佳路线，帮助规划者更容易理解和采纳。
* 导出功能，允许规划者将规划结果导出为文件，以便与其他部门或利益相关者共享。

1. 小组成员信息

组长：叶择锬

组员：李兵，杨子乐

2.项目描述

2.1问题描述

2.1.1拓扑排序原理说明

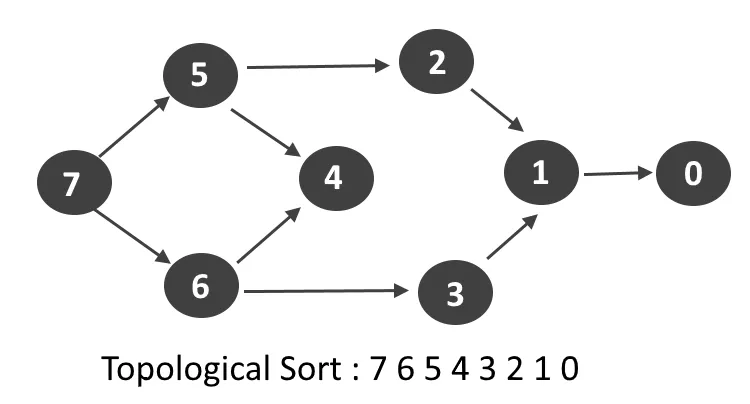
 拓扑排序是一种用于有向图（Directed Acyclic Graph，DAG）的线性排序算法，其原理是在有向无环图中，按照结点之间的关系将图中的节点线性排列，其底层实现方式主要有两种：Kahn 算法和深度优先搜索（DFS）算法。

图-3拓扑排序示意图

Kahn 算法是一种基于入度的拓扑排序算法，需要使用队列数据结构，算法主要的步骤如下：

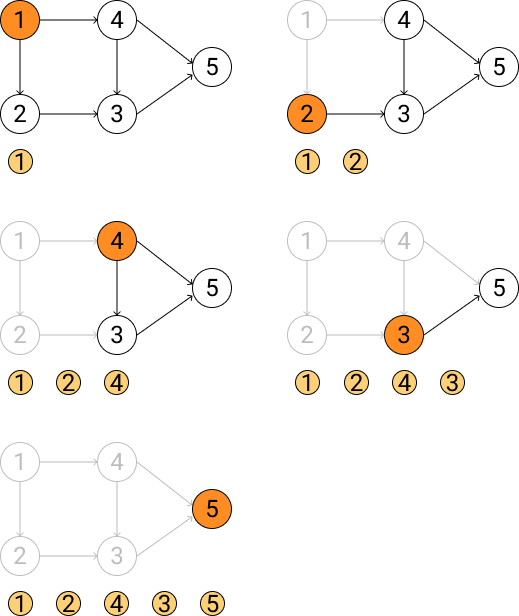
* 初始化一个队列，将所有入度为0的顶点加入队列中。
* 从队列中取出一个顶点，将其输出并将其所有邻接点的入度减1。
* 不断重复上述步骤，直到队列为空。如果最终输出的顶点数量等于图中的顶点数量，那么算法成功，否则，图中存在环。

图-4 Kahn 算法

深度优先搜索算法是一种用于图和树结构的搜索方法，其算法思想核心为通过深入尽可能多的路径，直到无法继续深入或找到目标，然后回溯并继续搜索其他路径，其算法思想可实现拓扑排序，算法主要步骤如下：

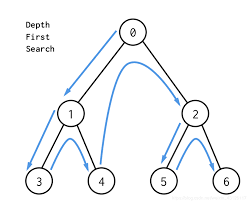
* 从图中选择一个未访问的顶点作为起始点，开始深度优先搜索。
* 在递归搜索的过程中，当遇到一个顶点的所有邻接点都已经被访问过，就将该顶点加入排序结果的列表中。
* 继续递归搜索其他未访问的顶点，直到所有顶点都被访问。

图-5 深度优先搜索算法思想

2.1.2具体应用场景

* 拓扑排序计算：基于项目提到的两种拓扑排序算法（Kahn算法和深度优先搜索算法），应用其中一种算法来计算站点之间的拓扑排序顺序。
* 站点依赖关系建模：将城市中的公共交通站点表示为图中的节点，站点之间的道路或具体线路抽象为图中的边。然后，利用拓扑排序原理来建模站点之间的依赖关系图，这一步骤可以通过图像的形式帮助规划者理解站点之间的联系，进而进一步规划合理科学的交通线路。
* 最佳路线规划：通过拓扑排序生成的站点顺序可以作为最佳路线规划的基础。规划者可以根据这个顺序确定最优的线路，以最小化乘客的等待时间和换乘次数。

2.2 系统功能性需求

2.2.1描述项目的主要功能

项目的主要功能是开发一个基于拓扑排序的交通和路线规划应用，具体包括以下方面：

1. 数据导入与处理：

* 该应用需要能够导入读写文件数据，如txt文件或其他格式的数据文件，以获取站点和站点之间的依赖关系信息。
* 用户可以输入站点数据，然后应用将根据这些数据计算出站点之间的依赖关系生成图，用以规划最佳的公交线路。

1. 拓扑排序计算：

* 根据导入的站点数据，应用将利用拓扑排序算法（如Kahn算法或深度优先搜索算法）来计算出站点之间的依赖关系。
* 拓扑排序的计算结果将作为后续站点依赖关系图的基础。

1. 可视化界面设计：

* 该应用需要设计一个用户友好的图形用户界面，使得城市规划者和交通规划者可以轻松上手使用。
* 界面设计应当直观明了，提供必要的交互操作，如“导入”、“导出”、“清空”等按钮。

1. 健壮性与错误处理：

* 在数据处理过程中，如果遇到输入错误或异常数据，应用需要进行错误处理，并通过弹出窗口等方式提示用户，以免用户误操作。

2.2.2说明图形界面的要求

对于应用的整体界面应该清晰、直观，导航简单。主界面应包括菜单栏、数据输入区、图形展示区和其他必要的功能区域。

* 菜单栏：应包括文件操作（如打开、保存、打印等）、编辑操作（如添加、删除、修改等）、视图操作（如放大、缩小、旋转等）、帮助、退出等菜单项。
* 图形展示区：用于展示和操作拓扑排序的图形界面，应具备放大、缩小、更换布局等功能。
* 导入/导出功能：在菜单栏或工具栏中，应提供导入和导出功能，允许用户导入站点数据和导出规划结果。导入和导出操作应该简单明了，最好能在几个步骤内完成。
* 数据可视化：在图形展示区，需要直观地展示站点之间的依赖关系。这样，用户可以一目了然地理解站点之间的关系。

2.2.3定义数据输入格式

统一规定数据输入的格式如下（注意：一行定义一个关系，西文下的尖括号）：<2,4><1,2><1,3><3,2>

使用有权图应该使用的输入格式为<a,b,2><b,c,4>

2.3 系统非功能性需求

2.3.1指定系统的运行平台

系统运行在Windous平台上。EXE文件是Windows可执行文件的标准扩展名，它们包含了Windows程序的可执行代码。

2.3.2选择开发工具和开发语言

选择PyCharm作为一款专门用于Python编程的IDE，提供了丰富的功能，此项目采用PyCharm这个开发工具和Python这门开发语言。程序使用PySide2库来创建UI，使用QT用户界面设计工具来创建UI，使前后端分离。

3.项目涉及的底层技术说明

本程序是一个基于Python的图形化应用程序，用于可视化和分析有向图的拓扑排序。程序的主要功能包括导入数据、生成图形、保存图形、显示帮助文档和退出应用。以下是对程序底层技术的详细说明：

3.1.UI界面：

程序使用PySide2库来创建用户界面（UI）。PySide2是Qt for Python的一个绑定库，用于构建跨平台的桌面应用程序。程序通过`loadUiType`函数加载了一个名为"busapp.ui"的UI文件，该文件定义了应用程序的用户界面。用户界面包括文本输入框、按钮、菜单栏等组件，以便用户可以输入数据、操作图形和导航应用。

该程序中使用动态加载UI，动态加载UI文件是一种在Python图形用户界面（GUI）应用程序中使用用户界面设计工具（如Qt Designer或Qt Creator）创建的UI文件的技术。这种技术允许开发者在运行时将UI文件加载到应用程序中，以便动态创建GUI界面。

动态加载UI文件相较于静态加载UI文件具备许多好处：

1）灵活性：

动态加载UI：动态加载UI文件使得在运行时可以根据需要更改或修改用户界面。这意味着可以根据特定条件或用户的选择来动态创建、显示或隐藏界面元素，从而实现更灵活的用户交互。例如，根据用户权限或配置文件，可以决定显示不同的界面元素。

静态加载UI：静态加载UI通常在设计时就已经定义好，很难在运行时进行修改或扩展，因此不太适合需要灵活性的情况。

2）分离设计和逻辑：

动态加载UI：使用动态加载UI的方法可以更好地将界面设计与应用程序的逻辑分离。设计人员可以专注于创建UI文件，而开发者可以专注于编写应用程序的后端逻辑。这种分离有助于团队协作，允许专业的设计人员和开发人员各自负责自己擅长的领域。

静态加载UI：静态加载UI通常需要在代码中硬编码UI元素，可能会导致界面设计与代码紧密耦合，使得维护和修改变得更加困难。

3） 动态生成界面：

动态加载UI：运行时动态生成界面的能力使得可以根据数据或其他条件创建用户界面的不同视图。例如，在一个数据驱动的应用程序中，可以根据数据库中的数据动态生成表单或列表。

静态加载UI：静态加载的UI通常是静态的，不太适合根据不同的数据或条件生成不同的界面。

4）减少冗余代码：

动态加载UI：动态加载UI通常可以减少代码重复。相同类型的界面元素只需要在UI文件中定义一次，并且可以在多个地方重复使用。这减少了维护和修改的工作量。

静态加载UI：静态加载UI可能需要在多个地方手动创建相同的界面元素，增加了代码的冗余。

动态加载UI使得开发者可以更灵活地设计和修改用户界面，而无需手动编写大量的UI代码，同时也有助于将界面设计与应用程序的逻辑分离。

3.2.图形生成：

程序使用NetworkX库来创建、分析和可视化有向图。NetworkX是一个用于处理复杂网络和图形分析的Python库。在程序中，首先从文本输入框中获取用户输入的图形数据，然后根据输入数据创建有向图。程序通过NetworkX的函数来检查图形是否存在循环依赖，如果存在，则会弹出警告提示框。接下来，程序使用spring\_layout布局算法来布置节点的位置，并使用Matplotlib库来绘制图形。程序可以生成并显示多个拓扑排序结果。

3.3.图形保存：

用户可以使用程序的菜单栏中的"保存图像"选项将生成的图形保存为PNG文件。程序使用Matplotlib的`savefig`函数将图形保存到用户指定的文件路径。

3.4.数据导入：

用户可以使用程序的菜单栏中的"导入数据"选项导入本地文本文件。程序使用Qt的文件对话框来让用户选择要导入的文件，然后将文件内容加载到文本输入框中。

3.5.帮助文档：

用户可以使用程序的菜单栏中的"帮助"选项查看帮助文档。该功能涉及Qt库中QMessageBox类的显示关于框（About Box）：用于显示应用程序的关于信息，通常包括应用程序名称、版本号、版权信息等。帮助文档包含了关于如何输入数据、生成结果以及导入数据的说明。

3.6.退出应用：

用户可以使用程序的菜单栏中的"退出"选项退出应用。该功能涉及Qt库中的QMessageBox类的显示提问框（Question Box）：用于向用户提出一个问题，通常需要用户选择一个选项。可以包括一个标题、一些文本消息和一组自定义按钮。程序会弹出确认退出的消息框，以确保用户的选择。

3.7.主函数：

程序的主函数在`if \_\_name\_\_ == '\_\_main\_\_':`部分，创建了一个Qt应用程序，初始化了主窗口（`GraphVisualizationApp`类的实例），并启动了应用程序的事件循环。

本程序使用了多个Python库和模块，包括PySide2、NetworkX、Matplotlib和Qt，来创建一个交互式的图形化应用程序。用户可以通过界面输入数据、生成图形、保存图形和查看帮助文档，从而更好地理解和分析有向图的拓扑排序。程序的模块化设计使其易于维护和扩展。

4.可行性分析

4.1 技术可行性

根据项目需要实现的主要功能，我们将从算法实现、开发工具和语言选择、可视化实现三个方面来分析实现该项目的技术可行性：

算法实现：

* Python是一种流行的编程语言，具有丰富的库和生态系统，适合进行复杂的数据处理和算法实现。使用Python语言进行算法实现具有高度的可行性，
* NetworkX库是一个Python库，专门用于图论和网络分析，NetworkX库中的拓扑排序算法基于深度优先搜索（DFS），在有向无环图（DAG）中具有良好的性能。这意味着可以有效地处理大规模的站点依赖关系图。

开发工具和语言：

* Python语言的易学性和社区支持使得开发过程更加高效。开发者可以利用Python的特性来快速迭代、调试和测试代码。
* 使用PySide6作为图形开发工具是合理的选择。PySide6是Python的官方Qt绑定，提供了强大的GUI开发功能，可以创建用户友好的图形用户界面。

可视化技术：

* PySide6支持丰富的可视化技术，您可以使用它来创建用户友好的图形用户界面，包括数据输入区和图形展示区，以便用户能够更好地进行交互，降低应用上手门槛。
* NetworkX库可以与Matplotlib库等可视化库结合使用，以图形方式展示拓扑排序的结果。这将允许用户在图形界面上直观地查看站点之间的依赖关系图。

在整体技术可行性方面，选择Python、PySide6、NetworkX库和Matplotlib库等开发工具和Python库的组合能够满足项目的需求。通过Python和这些库的易学性使开发过程更加高效，有助于项目按计划推进。

4.2 时间可行性

考虑到项目从9月4日开始，中期验收定于9月11日，总的验收在9月16日进行，只有约两周左右的开发时间，合理地拆分项目阶段尤为关键，以下从任务分解、开发周期两个角度展开分析：

任务分解

* 需求分析（9.4 - 9.6）： 了解项目需求，收集用户需求，并明确功能和界面设计要求。
* 设计（9.7 - 9.8）： 制定应用程序的整体设计，包括界面设计和数据处理流程。
* 编码（9.9 - 9.13）： 开始实际的编码工作，实现数据导入、拓扑排序算法、图形界面等功能。
* 测试和调试（9.14 - 9.15）： 对应用程序进行测试，解决潜在的问题和错误。
* 文档编写（9.16）： 编写项目文档，包括用户手册和README文档。

开发周期

* 周1（9.4 - 9.9）： 在第一周，需求分析、设计、算法选择和实现、界面设计、以及数据处理功能的实现都应该开始。这个阶段需要尽早完成核心算法的编写。
* 周2（9.10 - 9.16）： 第二周需要集中精力进行测试、修复和文档编写。确保用户界面的友好性和可视化功能的正确性。在项目结束前，进行总验收的准备工作。

5.项目成果展示

本项目旨在开发一个公交车站拓扑排序应用程序，该应用程序具有多项功能，包括导入导出文件、生成拓扑排序无权图、生成拓扑排序有权图、改变拓扑排序图的布局、帮助文档的查看、退出程序以及计算最短路径。以下是对这些功能的详细总结：

5.1. 导入导出文件功能：

实现了从文件导入数据（txt文件）的功能，允许用户将节点和连接信息从外部文件导入应用程序。允许用户将生成的拓扑排序图（png格式）和拓扑排序序列（txt文件）保存到文件，以便日后重复使用。

5.2. 生成拓扑排序无权图功能：

应用程序可以根据导入的数据生成拓扑排序图和拓扑排序序列，该图表示站点之间的连接关系。使用拓扑排序算法对站点进行排序，以确保不会存在环路。

5.3. 生成拓扑排序有权图功能：

除了无权图，应用程序还可以为站点之间的连接分配权重。用户可以指定不同站点之间的距离或时间等权重，以便进行更复杂的分析。用户通过自己输入距离可以计算出初始站点到其他任意站点的的最短路径。

5.4. 改变拓扑排序图的布局功能：

提供了可视化工具，允许用户更改拓扑排序图的布局，以更好地满足其可视化需求。用户可以自定义站点的位置，以使图形更具可读性。

5.5. 帮助文档的查看功能：

提供了帮助文档，帮助用户了解应用程序的功能和用法。用户可以随时查看帮助文档以获取指导。

5.6. 计算最短路径功能：

应用程序实现了计算任意两站点之间的最短路径的功能。用户可以选择初始站点，并查找此站点到任意站点的最短路径以获得站点之间的最佳路线。

5.7. 退出程序功能：

提供了退出程序的选项，以便用户可以在需要时安全地关闭应用程序。

本项目成功地实现了一个功能丰富的公交车站拓扑排序应用程序，为用户提供了多种有用的功能，以帮助他们分析和优化公交站点之间的连接关系。这个应用程序的潜力可以在未来得到更多的拓展和改进，以满足不断增长的需求。

6.项目总结与思考

在上述拓扑排序软件的实现过程中，涉及了多个功能和技术，以下是一些总结思考与感悟：

6.1. 软件设计的重要性： 在开发此类应用程序时，良好的软件设计是关键。清晰的架构、模块化的代码和良好的用户界面设计可以提高可维护性和用户体验。应该在初始设计阶段就把基本功能定好，后续增加一些拓展功能。在用户界面设计中，一开始并没有把拓展功能考虑进去，没有预先设定好功能按钮的预估空间摆放，导致后期重新更改用户界面，导致中间的一些bug产生，浪费了时间。

6.2. 数据处理与算法： 拓扑排序、最短路径等算法在公交车站排序应用程序中起到关键作用。正确实现和优化这些算法对于性能和功能的实现至关重要。

6.3. 文件导入导出： 数据的导入和导出功能允许用户在应用程序中轻松使用自己的数据。这是提高应用程序实用性的重要功能。

6.4. 可视化： 提供可视化工具使用户能够更好地理解数据和分析结果。可视化是复杂数据处理应用程序的一项关键要素。图片可视化是调用python提供的Matplotlib库，现代化的工具能够更好的提高编程效率，大大解放了人类劳动力。在这个项目中，我们使用AI工具协助书写代码和解决问题，感觉到工具越来越智能化，解决问题的效率大大提高。

6.5. 帮助文档与用户支持： 提供帮助文档和用户支持是确保用户能够正确使用应用程序的重要方面。它们有助于降低学习曲线并提高用户满意度。帮助文档能够为用户提供解释软件各种功能、操作、设置。例如我们使用pycharm的时候，许多库的资源也是通过研究软件的帮助文档得出一些库的用法与功能。

6.6. 扩展性： 软件应具有一定的扩展性，以便在未来能够添加更多功能和改进。这需要考虑到软件的模块化和可维护性。功能可以增加例如该站点某路车的到站时间预估，发车时间预估以及班车实时路况信息反馈。

6.7. 实践和学习： 开发这样的应用程序需要不断学习和实践。掌握新的技术和算法是不断改进和完善应用程序的关键。

总的来说，开发公交车站排序软件是一个有挑战性但也有很多收获的过程。通过这个项目，我们不仅能够提高编程技能，还能够为解决实际问题提供有用的工具。同时，这个项目也突出了软件开发中用户需求、算法和可视化的重要性，为今后的开发经验提供了宝贵的经验。

7.源代码

tuopu.ui

<?xml version="1.0" encoding="UTF-8"?>  
<ui version="4.0">  
 <class>MainWindow</class>  
 <widget class="QMainWindow" name="MainWindow">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>688</width>  
 <height>559</height>  
 </rect>  
 </property>  
 <property name="windowTitle">  
 <string>MainWindow</string>  
 </property>  
 <widget class="QWidget" name="centralwidget">  
 <layout class="QFormLayout" name="formLayout">  
 <item row="0" column="0">  
 <layout class="QHBoxLayout" name="horizontalLayout\_2">  
 <item>  
 <layout class="QVBoxLayout" name="verticalLayout">  
 <item>  
 <widget class="QPlainTextEdit" name="plainTextEdit">  
 <property name="placeholderText">  
 <string>若生成无权图，请输入&lt;a,b&gt;,&lt;c,d&gt;类型的数据；若生成有权图，请输入&lt;a,b,3&gt;类型的数据，并在最下方框内输入起始点</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QComboBox" name="comboBox">  
 <property name="editable">  
 <bool>true</bool>  
 </property>  
 <item>  
 <property name="text">  
 <string>Spectral Layout</string>  
 </property>  
 </item>  
 <item>  
 <property name="text">  
 <string>Shell Layout</string>  
 </property>  
 </item>  
 <item>  
 <property name="text">  
 <string>Circular Layout</string>  
 </property>  
 </item>  
 <item>  
 <property name="text">  
 <string>Spring Layout</string>  
 </property>  
 </item>  
 </widget>  
 </item>  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout" stretch="0,0">  
 <item>  
 <widget class="QPushButton" name="pushButton">  
 <property name="text">  
 <string>生成有权图</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QPushButton" name="pushButton\_2">  
 <property name="text">  
 <string>生成无权图</string>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 <item>  
 <widget class="QTextEdit" name="textEdit"/>  
 </item>  
 </layout>  
 </item>  
 <item>  
 <widget class="QWidget" name="widget" native="true">  
 <property name="sizePolicy">  
 <sizepolicy hsizetype="Expanding" vsizetype="Expanding">  
 <horstretch>0</horstretch>  
 <verstretch>0</verstretch>  
 </sizepolicy>  
 </property>  
 <property name="minimumSize">  
 <size>  
 <width>10</width>  
 <height>10</height>  
 </size>  
 </property>  
 <property name="maximumSize">  
 <size>  
 <width>16777212</width>  
 <height>16777214</height>  
 </size>  
 </property>  
 <property name="layoutDirection">  
 <enum>Qt::RightToLeft</enum>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 </layout>  
 </widget>  
 <widget class="QMenuBar" name="menubar">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>688</width>  
 <height>22</height>  
 </rect>  
 </property>  
 <widget class="QMenu" name="menuFile">  
 <property name="title">  
 <string>File</string>  
 </property>  
 <widget class="QMenu" name="menuSave\_as">  
 <property name="title">  
 <string>Save as</string>  
 </property>  
 <addaction name="actionfile"/>  
 <addaction name="actionGraph"/>  
 </widget>  
 <addaction name="actionOpen"/>  
 <addaction name="menuSave\_as"/>  
 </widget>  
 <widget class="QMenu" name="menuGenerate">  
 <property name="title">  
 <string>Generate</string>  
 </property>  
 <addaction name="separator"/>  
 <addaction name="actionUnauthorized\_Graph"/>  
 <addaction name="actionAuthorized\_Graph"/>  
 </widget>  
 <widget class="QMenu" name="menuHelp">  
 <property name="title">  
 <string>Help</string>  
 </property>  
 <addaction name="actionReadme"/>  
 </widget>  
 <widget class="QMenu" name="menuExit">  
 <property name="title">  
 <string>Exit</string>  
 </property>  
 <addaction name="actionProgram\_Exit"/>  
 </widget>  
 <addaction name="menuFile"/>  
 <addaction name="menuGenerate"/>  
 <addaction name="menuHelp"/>  
 <addaction name="menuExit"/>  
 </widget>  
 <action name="actionOpen">  
 <property name="text">  
 <string>Open</string>  
 </property>  
 </action>  
 <action name="actionfile">  
 <property name="text">  
 <string>file</string>  
 </property>  
 </action>  
 <action name="actionUnauthorized\_Graph">  
 <property name="text">  
 <string>Unauthorized Graph</string>  
 </property>  
 </action>  
 <action name="actionAuthorized\_Graph">  
 <property name="text">  
 <string>Authorized Graph</string>  
 </property>  
 </action>  
 <action name="actionGraph">  
 <property name="text">  
 <string>Graph</string>  
 </property>  
 </action>  
 <action name="actionReadme">  
 <property name="text">  
 <string>README</string>  
 </property>  
 </action>  
 <action name="actionProgram\_Exit">  
 <property name="text">  
 <string>Program Exit</string>  
 </property>  
 </action>  
 </widget>  
 <resources/>  
 <connections/>  
</ui>

busapp.py

# 导入必要的库  
import numpy as np  
import sys  
import threading  
import networkx as nx  
import matplotlib.pyplot as plt  
import re  
import warnings  
import os  
  
warnings.filterwarnings("ignore", category=UserWarning)  
  
# 导入PySide2库中的相关模块  
from PySide2.QtWidgets import QApplication, QMainWindow, QFileDialog, QMessageBox, QTextEdit  
from PySide2.QtUiTools import loadUiType # 用于加载UI文件的函数  
from PySide2.QtCore import QFile, QTextStream, QSize  
from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas  
  
  
# 创建一个Qt应用程序  
class GraphVisualizationApp(QMainWindow):  
  
 def setWindow(self):  
 width = 965  
 height = 800  
 self.resize(QSize(width, height))  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 # 加载UI文件  
 self.ui = loadUiType("tuopu.ui")[0]() # 这里加载了一个UI文件  
 self.ui.setupUi(self)  
 self.setWindow()  
 # 初始化图像生成状态变量  
 self.graphGenerated = False  
  
 self.G = None # 用于存储图的变量  
  
 plt.rcParams['font.sans-serif'] = ['Microsoft YaHei'] # 使用微软雅黑或其他中文字体  
 # 创建matplotlib视图和图形canvas  
 self.graphView = plt.figure() # 创建Matplotlib视图  
 self.graphCanvas = FigureCanvas(self.graphView) # 创建Matplotlib图形的Canvas  
  
 # 将canvas添加到ui布局中  
 self.ui.horizontalLayout\_2.addWidget(self.graphCanvas)  
  
 # 创建QPlainTextEdit对象，用于文本输入和显示  
 self.textEdit = QTextEdit()  
 self.textEdit.setReadOnly(True)  
  
 # 链接ui菜单栏中的各个操作到相应的方法  
 self.ui.actionOpen.triggered.connect(self.importData) # 导入数据  
 self.ui.actionGraph.triggered.connect(self.saveGraph) # 保存图像  
 self.ui.actionUnauthorized\_Graph.triggered.connect(self.generateGraph) # 生成图  
 self.ui.actionReadme.triggered.connect(self.help) # 显示帮助窗口  
 self.ui.actionProgram\_Exit.triggered.connect(self.close\_window) # 退出应用  
 self.ui.actionfile.triggered.connect(self.exportResults) # 导出排序结果  
 self.ui.actionAuthorized\_Graph.triggered.connect(self.weightedGraph) # 生成带权图数据  
 self.ui.pushButton.clicked.connect(self.weightedGraph) # 生成有权图  
 self.ui.pushButton\_2.clicked.connect(self.generateGraph) # 生成无权图  
  
 def generateGraph(self):  
 # 获取文本框中的输入，每行定义一个关系，格式为 "<a,b>"  
 input\_text = self.ui.plainTextEdit.toPlainText()  
  
 # 使用正则表达式匹配输入格式  
 pattern = r'<(\w+),(\w+)>|《(\w+),(\w+)》'  
 matches = re.findall(pattern, input\_text)  
  
 edges = []  
  
 # 判断是否匹配到任何数据  
 if not matches:  
 QMessageBox.warning(self, "错误", "请添加正确格式的数据，例如：<a,b>", QMessageBox.Ok)  
 return  
  
 # 遍历匹配结果  
 for match in matches:  
 a, b, c, d = match  
 if a: # 使用尖括号的情况  
 edges.append((a.strip(), b.strip()))  
 elif c: # 使用双尖括号的情况  
 edges.append((c.strip(), d.strip()))  
 else:  
 # 输入不符合规范，显示错误提示  
 QMessageBox.warning(self, "错误", "请添加正确格式的数据，例如：<a,b>", QMessageBox.Ok)  
 return  
  
 # 创建有向图  
 G = nx.DiGraph()  
 G.add\_edges\_from(edges)  
  
 # 检查是否存在依赖关系  
 if not nx.is\_directed\_acyclic\_graph(G):  
 # 存在依赖关系，显示错误提示  
 QMessageBox.warning(self, "错误", "输入的数据存在依赖关系", QMessageBox.Ok)  
 return  
  
 # 根据用户选择的布局来计算节点的位置  
 layout\_option = self.ui.comboBox.currentText()  
 if layout\_option == "Spectral Layout":  
 pos = nx.spectral\_layout(G)  
 elif layout\_option == "Shell Layout":  
 pos = nx.shell\_layout(G)  
 elif layout\_option == "Circular Layout":  
 pos = nx.circular\_layout(G)  
 else:  
 pos = nx.spring\_layout(G, seed=10) # 默认使用spring\_layout布局  
  
 # 计算所有拓扑排序  
 all\_topo\_orders = list(nx.all\_topological\_sorts(G))  
 if not all\_topo\_orders:  
 print("无法进行拓扑排序，图中存在循环依赖。")  
 return []  
  
 # 清空Matplotlib视图并绘制拓扑排序关系图  
 plt.clf()  
 first\_topo\_order = all\_topo\_orders[0]  
 nx.draw(G, pos, with\_labels=True, node\_size=500, node\_color="skyblue", font\_size=10)  
 plt.title("拓扑排序")  
 self.graphCanvas.draw()  
 self.graphGenerated = True # 标记图像已生成  
  
 # 打印所有拓扑排序结果  
 print("所有拓扑排序结果:")  
 for topo\_order in all\_topo\_orders:  
 print(topo\_order)  
 return all\_topo\_orders  
  
 # Floyd最短路径算法  
 def floyd\_shortest\_paths(self, start\_vertex):  
 if not self.graphGenerated:  
 QMessageBox.warning(self, "错误", "您未生成任何图像", QMessageBox.Ok)  
 return  
  
 if self.G is None:  
 QMessageBox.warning(self, "错误", "带权图为空，请先导入带权图数据", QMessageBox.Ok)  
 return  
  
 # 获取所有顶点列表  
 nodes = list(self.G.nodes())  
  
 # 创建节点到索引的映射和反向映射  
 node\_to\_index = {node: i for i, node in enumerate(nodes)}  
 index\_to\_node = {i: node for i, node in enumerate(nodes)}  
  
 # 初始化距离矩阵，用inf表示不可达  
 num\_nodes = len(nodes)  
 distance\_matrix = np.inf \* np.ones((num\_nodes, num\_nodes))  
  
 # 将直接可达的顶点间距离填入距离矩阵  
 for edge in self.G.edges(data=True):  
 a, b, weight = edge[0], edge[1], edge[2]["weight"]  
 distance\_matrix[node\_to\_index[a]][node\_to\_index[b]] = weight  
  
 # Floyd算法求最短路径  
 for k in range(num\_nodes):  
 for i in range(num\_nodes):  
 for j in range(num\_nodes):  
 if distance\_matrix[i][k] + distance\_matrix[k][j] < distance\_matrix[i][j]:  
 distance\_matrix[i][j] = distance\_matrix[i][k] + distance\_matrix[k][j]  
  
 # 构建最短路径结果字符串  
 result\_text = "从顶点 '{}' 出发的最短路径：\n".format(start\_vertex)  
 for i in range(num\_nodes):  
 if index\_to\_node[i] != start\_vertex:  
 shortest\_path = distance\_matrix[node\_to\_index[start\_vertex]][i]  
 if shortest\_path == np.inf:  
 result\_text += "顶点 '{}' 不可达到顶点 '{}'。\n".format(start\_vertex, index\_to\_node[i])  
 else:  
 result\_text += "到顶点 '{}' 的最短路径为：{}\n".format(index\_to\_node[i], shortest\_path)  
  
 self.ui.textEdit.setText(result\_text) # 打印最短路径到textEdit框中  
  
 def weightedGraph(self):  
 # 获取文本框中的输入，每行定义一个带权边，格式为 "<a,b,1>"  
 # a和b为顶点，c为ab边的权重  
 input\_text = self.ui.plainTextEdit.toPlainText()  
 weighted\_edges = []  
  
 # 使用正则表达式匹配输入格式  
 pattern = r'<(\w+),(\w+),(\d+)>|《(\w+),(\w+),(\d+)》'  
 matches = re.findall(pattern, input\_text)  
  
 # 判断是否匹配到任何数据  
 if not matches:  
 QMessageBox.warning(self, "错误", "请添加正确格式的数据，例如：<a,b,1>", QMessageBox.Ok)  
 return  
  
 # 遍历匹配结果  
 for match in matches:  
 a, b, c, d, e, f = match  
 if a: # 使用尖括号的情况  
 weighted\_edges.append((a.strip(), b.strip(), int(c)))  
 elif d: # 使用双尖括号的情况  
 weighted\_edges.append((d.strip(), e.strip(), int(f)))  
 else:  
 # 输入不符合规范，显示错误提示  
 QMessageBox.warning(self, "错误", "请添加正确格式的数据，例如：<a,b,1>", QMessageBox.Ok)  
 return  
  
 # 创建带权有向图  
 self.G = nx.DiGraph()  
 for edge in weighted\_edges:  
 a, b, weight = edge  
 self.G.add\_edge(a, b, weight=weight)  
  
 layout\_option = self.ui.comboBox.currentText()  
 if layout\_option == "Spectral Layout":  
 pos = nx.spectral\_layout(self.G)  
 elif layout\_option == "Kamada-Kawai Layout":  
 pos = nx.kamada\_kawai\_layout(self.G)  
 elif layout\_option == "Circular Layout":  
 pos = nx.circular\_layout(self.G)  
 else:  
 pos = nx.spring\_layout(self.G, seed=10) # 默认使用spring\_layout布局  
  
 plt.clf()  
 labels = {(a, b): weight for a, b, weight in self.G.edges(data="weight")}  
 edge\_weights = [weight for a, b, weight in self.G.edges(data="weight")]  
  
 nx.draw(self.G, pos, with\_labels=True, node\_size=500, node\_color="skyblue", font\_size=10)  
 nx.draw\_networkx\_edge\_labels(self.G, pos, edge\_labels=labels, font\_size=8, rotate=False)  
  
 plt.title("带权图")  
 self.graphCanvas.draw()  
 self.graphGenerated = True  
  
 self.generateOptimalRoute()  
  
 # 生成最优路线  
 def generateOptimalRoute(self):  
 # 在这里实现生成最优路线的功能  
 # 使用图算法（如最短路径算法）计算最优路线  
 # 这里以最短路径算法Dijkstra为例  
 # 假设起始点为start\_vertex，可以根据实际情况修改  
 start\_vertex = self.ui.textEdit.toPlainText()  
 if not self.graphGenerated:  
 QMessageBox.warning(self, "错误", "您未生成任何图像", QMessageBox.Ok)  
 return  
  
 if not start\_vertex:  
 QMessageBox.warning(self, "错误", "请输入起始点", QMessageBox.Ok)  
  
 if self.G is None:  
 QMessageBox.warning(self, "错误", "带权图为空，请先导入带权图数据", QMessageBox.Ok)  
 return  
  
 self.floyd\_shortest\_paths(start\_vertex)  
  
 # 保存生成的图像  
 def saveGraph(self):  
 if not self.graphGenerated:  
 # 如果没有生成图像，显示弹出窗口提示  
 QMessageBox.warning(self, "错误", "您未生成任何图像", QMessageBox.Ok)  
 return  
  
 # 弹出文件对话框，允许用户选择保存图像的路径和文件名  
 file\_dialog = QFileDialog()  
 file\_path, \_ = file\_dialog.getSaveFileName(self, "保存图像", "", "PNG Files (\*.png);;All Files (\*)")  
 if file\_path:  
 # 保存图像到指定路径  
 self.graphView.savefig(file\_path, format='png')  
  
 # 导出结果到文本文件  
 def exportResults(self, result\_text):  
 if not self.graphGenerated:  
 # 如果没有生成图像，显示弹出窗口提示  
 QMessageBox.warning(self, "错误", "您未生成任何图像", QMessageBox.Ok)  
 return  
  
 all\_topo\_orders = self.generateGraph()  
 result\_text = "\n".join(["->".join(order) for order in all\_topo\_orders])  
 # 弹出文件对话框，允许用户选择保存结果的路径和文件名  
 file\_dialog = QFileDialog()  
 file\_path, \_ = file\_dialog.getSaveFileName(self, "保存结果", "", "Text Files (\*.txt);;All Files (\*)")  
 if file\_path:  
 try:  
 # 打开文件并将结果写入  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(result\_text)  
  
 QMessageBox.information(self, "提示", "结果已成功导出到文件：" + file\_path, QMessageBox.Ok)  
 except Exception as e:  
 QMessageBox.warning(self, "错误", "导出结果时出现错误：" + str(e), QMessageBox.Ok)  
  
 # 导入数据文件  
 def importData(self):  
 # 弹出文件对话框，允许用户自行选择要导入的本地文本文件  
 file\_dialog = QFileDialog()  
 file\_path, \_ = file\_dialog.getOpenFileName(self, "导入数据", "", "Text Files (\*.txt);;All Files (\*)")  
 # 如果用户选择了文件，就将文件内容加载到文本编辑框  
 if file\_path:  
 # 读取文件内容  
 file = QFile(file\_path)  
 if file.open(QFile.ReadOnly | QFile.Text):  
 stream = QTextStream(file)  
 self.ui.plainTextEdit.setPlainText(stream.readAll())  
 file.close()  
  
 # 关闭应用程序  
 def close\_window(self):  
 # 创建一个确认退出的消息框  
 reply = QMessageBox.question(self, "确认退出", "确定要退出吗？", QMessageBox.Yes | QMessageBox.No,  
 QMessageBox.No)  
  
 # 如果用户选择是（Yes），则关闭窗口  
 if reply == QMessageBox.Yes:  
 self.close()  
   
 # 在GraphVisualizationApp类中添加新的help方法  
 def help(self):  
 # 定义README文件的路径  
 readme\_file\_path = "README.txt"  
  
 # 检查README文件是否存在  
 if os.path.exists(readme\_file\_path):  
 try:  
 os.system("start notepad " + readme\_file\_path)  
 except Exception as e:  
 QMessageBox.warning(self, "错误", "无法打开README文件：" + str(e), QMessageBox.Ok)  
 else:  
 QMessageBox.warning(self, "错误", "README文件不存在,请检查当前目录是否有README.txt文件", QMessageBox.Ok)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 ex = GraphVisualizationApp()  
 ex.show()  
 sys.exit(app.exec\_())