

**2025年春季学期  
计算学部《软件工程》课程**

**Lab 1实验报告**

|  |  |  |
| --- | --- | --- |
| **姓名** | **班级/学号** | **联系方式** |
| 刘吉涛 | 2203202/2022112961 | 15828788569 |

**目 录**

[1 实验要求 1](#_Toc166175466)

[2 待求解问题描述 1](#_Toc166175467)

[3 算法与数据结构设计 1](#_Toc166175468)

[3.1 设计思路与算法流程图 1](#_Toc166175469)

[3.2 数据结构设计 1](#_Toc166175470)

[3.3 算法时间复杂度分析 1](#_Toc166175471)

[4 实验与测试 1](#_Toc166175472)

[4.1 读取文本文件并展示有向图 2](#_Toc166175473)

[4.2 查询桥接词 2](#_Toc166175474)

[4.3 根据桥接词生成新文本 2](#_Toc166175475)

[4.4 计算最短路径 3](#_Toc166175476)

[4.5 随机游走 3](#_Toc166175477)

[5 编程语言与开发环境 3](#_Toc166175478)

[6 结对编程 3](#_Toc166175479)

[6.1 分组依据 3](#_Toc166175480)

[6.2 角色切换与任务分工 3](#_Toc166175481)

[6.3 工作照片 4](#_Toc166175482)

[6.4 工作日志 4](#_Toc166175483)

[7 Git操作过程 4](#_Toc166175484)

[7.1 实验场景(1)：仓库创建与提交 4](#_Toc166175485)

[7.2 实验场景(2)：分支管理 4](#_Toc166175486)

[8 在IDE中使用Git Plugin 4](#_Toc166175487)

[9 小结 4](#_Toc166175488)

[文档全部完成之后，请在上述区域点击右键，选择“更新域”，在打开的对话框中选择“更新整个目录”]

# 实验要求

实验内容 1：基于大模型的编程

– 熟悉面向对象的编程；

– 掌握利用大模型辅助编程的方式；

– 实验同大模型的“结对编程”。

实验内容 2：Git 实战

– 熟练掌握 Git 的基本指令和分支管理指令；

– 掌握 Git 支持软件配置管理的核心机理；

– 在实践项目中使用 Git /Github 管理自己的项目源代码。

# 待求解问题描述

试验要求从文本文件中读取英文文本数据，生成有向图并进行多种操作。具体描述如下：

– 输入数据：一个包含英文文本的文件，文本中的换行符、标点符号均视为空格，非

字母字符被忽略。

– 输出数据：根据输入文本生成的有向图，以及在图上进行操作后的结果，如桥接词

查询结果、新文本生成结果、最短路径计算结果、PageRank 值计算结果和随机游走结

果等。为方便操作和用户友好，使用图形化操作界面。

– 约束条件：PR 计算(d 取值 0.85)，其中出度为 0 的节点需要将 PR 值均分给其他 节点。

# 算法与数据结构设计

## 设计思路与算法流程图

给出文字描述的算法设计思路。

使用大一高级语言编程课里所学知识，用流程图的方式描述算法设计思路，让读者清晰理解。如果需要，可针对每一个模块分别绘制流程（根据文本生成图、展示图、查询桥接词、根据桥接词生成新文本、计算最短路径、计算PR值、随机游走）。

程序整体流程如下：

1、用户选择或输入文本文件路径。

2、程序读取文本文件，预处理文本（去除标点、转小写等）并生成有向图。

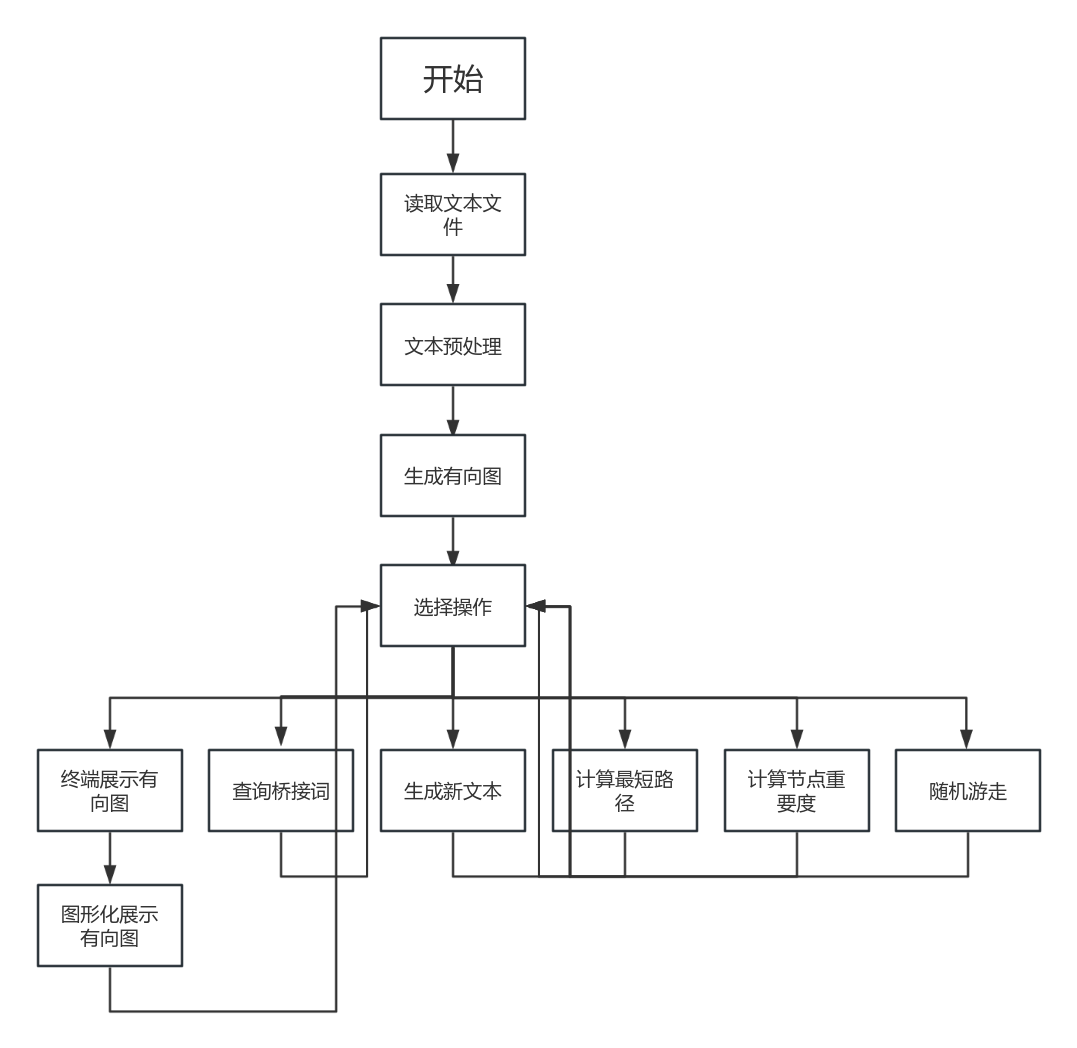
3、通过图形化界面或命令行展示有向图。

4、提供多种功能操作供用户选择，如查询桥接词、生成新文本、计算最短路径、计算

PageRank 值和随机游走等。

1. 输出相应操作结果。

流程图：



## 数据结构设计

使用大二期间数据结构的相关知识，给出算法中所用的每个数据结构的定义。至少需定义本题中要求生成的有向图的数据结构。

在代码中，主要使用了以下几种数据结构来实现有向图的构建和操作。以下是这些数据结构的详细定义：

1. 有向图的数据结构

有向图使用邻接表表示，具体实现如下：

class DirectedGraph:

def \_\_init\_\_(self):

self.graph = defaultdict(lambda: defaultdict(int))

self.fig = None

self.ax = None

self.pos = None

self.G = None

self.dragging\_node = None

self.drag\_start\_pos = None

self.node\_size = 2000 统一节点大小

**`self.graph`**

- 类型：`defaultdict(lambda: defaultdict(int))`

- 定义：这是一个嵌套的字典，外层字典的键是节点，值是另一个字典。内层字典的键是目标节点，值是边的权重。

- 用途：存储有向图的邻接表表示。例如，`self.graph[u][v]` 表示从节点 `u` 到节点 `v` 的边的权重。

**`self.fig` 和 `self.ax`**

- 类型：`matplotlib.figure.Figure` 和 `matplotlib.axes.Axes`

- 定义：`self.fig` 是 Matplotlib 的图形对象，`self.ax` 是图形中的坐标轴对象。

- 用途：用于绘制和显示有向图的图形化界面。

**`self.pos`**

- 类型：`dict`

- 定义：字典，键是节点，值是节点的位置坐标（`x, y`）。

- 用途：存储每个节点在图形化界面中的位置。

**`self.G`**

- 类型：`networkx.DiGraph`

- 定义：NetworkX 的有向图对象。

- 用途：用于构建和操作有向图，支持各种图算法。

**`self.dragging\_node` 和 `self.drag\_start\_pos`**

- 类型：`str` 和 `tuple`

- 定义：`self.dragging\_node` 是当前正在拖动的节点，`self.drag\_start\_pos` 是拖动开始时的节点位置。

- 用途：支持节点的拖动功能，用于交互式图形化界面。

**`self.node\_size`**

- 类型：`int`

- 定义：节点的大小。

- 用途：在绘制图形时，设置节点的大小。

**2. 其他辅助数据结构**

**`defaultdict`**

- 类型：`collections.defaultdict`

- 定义：`defaultdict` 是一个字典子类，它提供了一个工厂函数，用于自动初始化默认值。

- 用途：在代码中，`defaultdict` 用于初始化邻接表，避免手动检查键是否存在。

**`deque`**

- 类型：`collections.deque`

- 定义：`deque` 是一个双端队列，支持从两端高效地添加和删除元素。

- 用途：在 `calcShortestPath` 方法中，`deque` 用于实现广度优先搜索（BFS）算法。

**`numpy`**

- 类型：`numpy` 是一个用于科学计算的库，支持高效的数组操作。

- 定义：在代码中，`numpy` 用于计算节点位置和绘制箭头。

- 用途：在 `\_draw\_edges\_with\_arrows` 方法中，`numpy` 用于计算节点之间的距离和缩进。

**3. 具体数据结构的定义**

**`self.graph` 的具体定义**

self.graph = defaultdict(lambda: defaultdict(int))

- 外层字典：`defaultdict`，键是节点（`str`），值是另一个字典。

- 内层字典：`defaultdict(int)`，键是目标节点（`str`），值是边的权重（`int`）。

**`self.pos` 的具体定义**

self.pos = nx.spring\_layout(self.G, k=1, iterations=50)

- 类型：`dict`

- 键：节点（`str`）

- 值：节点的位置坐标（`tuple`，包含 `x` 和 `y` 坐标）

**`self.G` 的具体定义**

self.G = nx.DiGraph()

- 类型：`networkx.DiGraph`

- 用途：存储有向图的结构，支持各种图算法。

**`self.node\_collection` 的具体定义**

self.node\_collection = nx.draw\_networkx\_nodes(

self.G, self.pos,

node\_color='lightblue',

node\_size=self.node\_size,

alpha=0.9,

ax=self.ax

)

- 类型：`matplotlib.collections.PathCollection`

- 用途：存储绘制的节点集合，用于图形化显示。

## 算法时间复杂度分析

可针对每一个模块分别分析（根据文本生成图、展示图、查询桥接词、根据桥接词生成新文本、计算最短路径、计算PR值、随机游走）。

有向图生成： 总时间复杂度：O()（主要由布局计算决定）。

桥接词查询：时间复杂度为 O(k)，其中 k 为图中边的数量。需遍历指定节点的出边，

查找满足条件的桥接词。

新文本生成：时间复杂度为 O(l)，其中 l 为输入文本的单词数量。需遍历输入文本单

词，查询桥接词并生成新文本。

最短路径计算（BFS）：时间复杂度为 O(n + m)，其中 n 为节点数，m 为边数。需遍

历所有节点和边进行 BFS 搜索功能。

PageRank 计算：时间复杂度为 O(n × iter)，其中 iter 为迭代次数。需多次遍历所有

节点进行 PR 值更新。

随机游走：时间复杂度为 O(w)，其中 w 为游走路径长度。需根据游走规则逐步生成

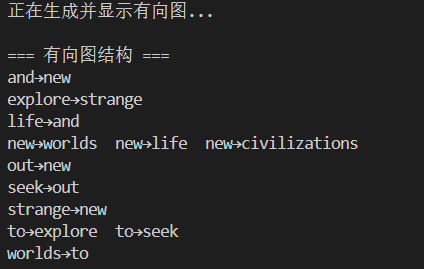
路径。

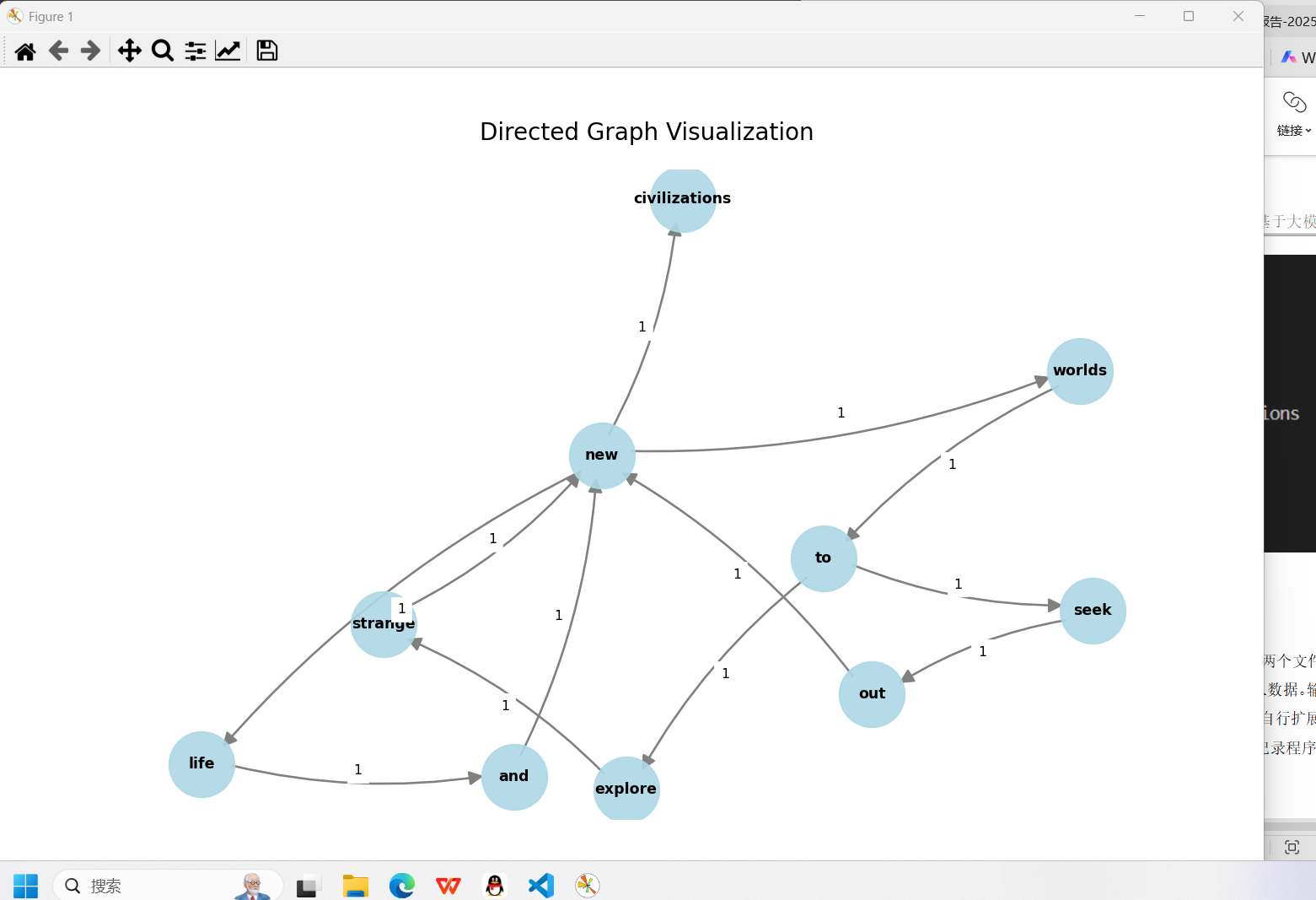
## 算法代码的生成

要求在显示有向图图形化界面的同时通过自定义的格式在 CLI（命令行界面）上进行 展示生成的有向图，要求格式清晰，易于理解。

如：A→B A→C

B→D





# 实验与测试

利用提供的Easy Test.txt文件和Cursed Be The Treasure.txt两个文件进行测试。

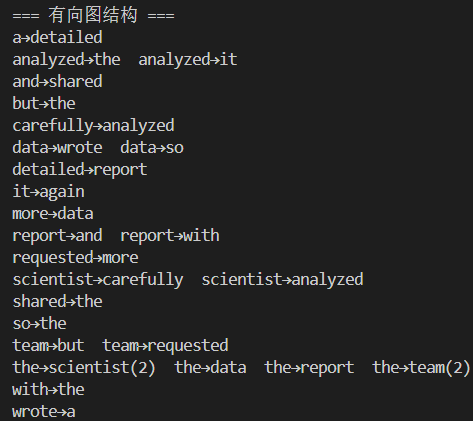
针对在有向图上操作的每项功能，为其设计各种可能的输入数据。输入数据的数量不限，以测试程序的充分性为评判标准（下面各节中的表格的行数请自行扩展）。

记录程序的输出结果，判断输出结果是否与期望一致，并记录程序运行截图。**下面各项只需要填写针对Easy Test.txt的实验结果。**

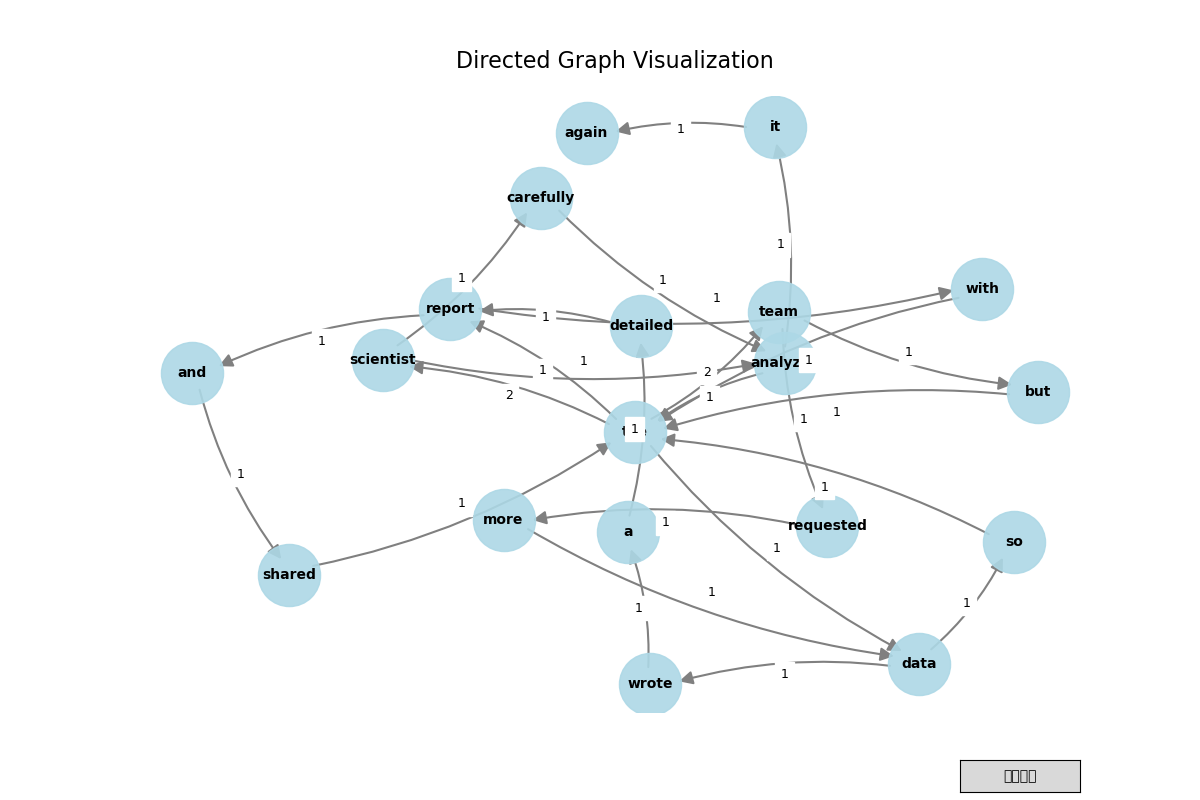
## 读取文本文件并展示有向图

文本文件中包含的内容：The scientist carefully analyzed the data, wrote a detailed report, and shared the report with the team, but the team requested more data, so the scientist analyzed it again.

期望生成的图（手工计算得到）：



程序实际生成的图：

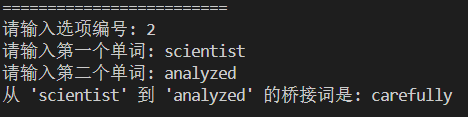


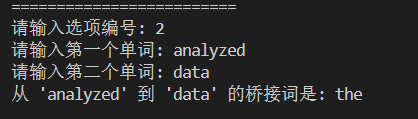
二者一致

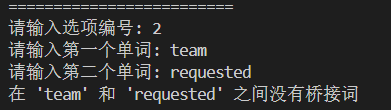
## 查询桥接词

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 输入（2个单词） | 期望输出 | 实际输出 | 是否正确 |
| 1 | scientist, analyzed | 从'scientist'到'analyzed'的桥接词是: carefully | 从'scientist'到'analyzed' 的桥接词是: carefully | 正确 |
| 2 | analyzed, data | 从 'analyzed' 到 'data'的桥接词是: the | 从 'analyzed' 到 'data'的桥接词是: the | 正确 |
| 3 | team, requested | 在 'team' 和 'requested'  之间没有桥接词 | 在 'team' 和 'requested'  之间没有桥接词 | 正确 |

给出实际运行得到结果的界面截图。



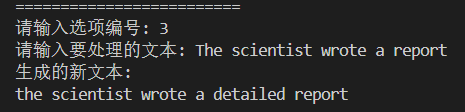


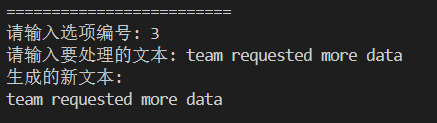


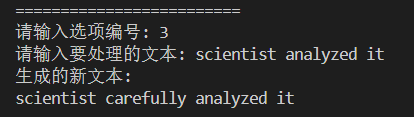
## 根据桥接词生成新文本

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 输入（一行文本） | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | The scientist wrote  a report | the scientist wrote a  detailed report | the scientist wrote a  detailed report | 正确 |
| 2 | team requested  more data | team requested more data | team requested more data | 正确 |
| 3 | scientist analyzed  it | scientist carefully  analyzed it | scientist carefully  analyzed it | 正确 |

给出实际运行得到结果的界面截图。



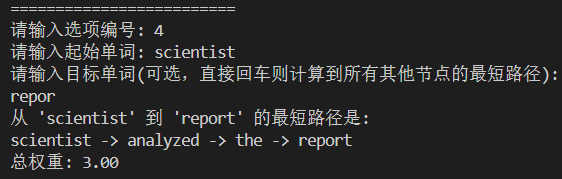


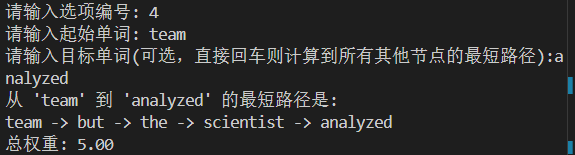


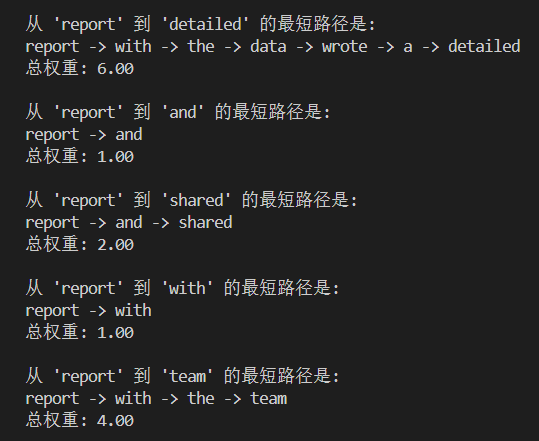
## 计算最短路径

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 输入（两个单词、或一个单词） | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | scientist, report | 从 'scientist' 到 'report'  的最短路径是:  scientist -> analyzed ->  the -> report  总权重: 3.00 | 从 'scientist' 到 'report'  的最短路径是:  scientist -> analyzed ->  the -> report  总权重: 3.00 | 正确 |
| 2 | team, analyzed | 从 'team' 到 'analyzed'  的最短路径是:  team -> but -> the ->  scientist -> analyzed  总权重: 5.00 | 从 'team' 到 'analyzed'  的最短路径是:  team -> but -> the ->  scientist -> analyzed  总权重: 5.00 | 正确 |
| 3 | report | 从 'report' 出发的所有  最短路径：  到 'and':  路径: report -> and  总权重: 1.00  到 'with':  路径: report -> with  总权重: 1.00  ……略 | 从 'report' 出发的所有  最短路径：  到 'and':  路径: report -> and  总权重: 1.00  到 'with':  路径: report -> with  总权重: 1.00  ……略 |  |

给出实际运行得到结果的界面截图。







## 计算PageRank值

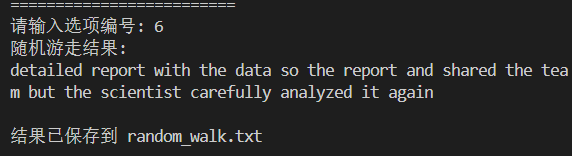
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 单词 | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | the | 0.1707 | 0.1707 | 正确 |
| 2 | report | 0.0727 | 0.0727 | 正确 |
| 3 | data | 0.0670 | 0.0670 | 正确 |

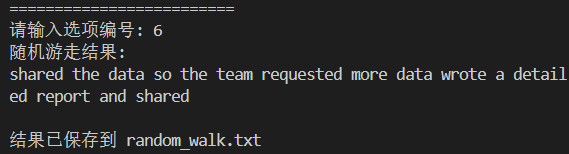
## 随机游走

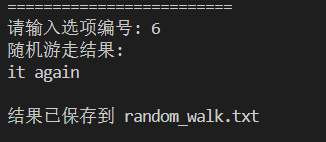
该功能无输入，让你的程序执行多次，分别记录结果。

|  |  |  |
| --- | --- | --- |
| 序号 | 实际输出 | 程序运行是否正确 |
| 1 | detailed report with the data so the report and shared the team but the scientist carefully analyzed it again | 正确 |
| 2 | shared the data so the team requested more data wrote a detailed report and shared | 正确 |
| 3 | it again | 正确 |

给出实际运行得到结果的界面截图。







# **编程语言与开发环境**

Python3 版本、IDE（vscode）；

采用的大模型：kimi。

# **Git操作过程**

## **实验场景(1)：仓库创建与提交**

R0：查看工作区、暂存区、git 仓库的状态

git status

R1：本地初始化一个 git 仓库

git init

R2：将项目文件加入 git 管理并提交

git add .

git commit -m "Initial commit"

R3：修改文件后查看状态和修改内容

使用文本编辑器修改文件

git status

git diff

R4：重新提交修改

git add .

git commit -m "Updated files"

R5：再次修改文件并提交

使用文本编辑器再次修改文件

git add .

git commit -m "Another update"

R6：撤销最后一次提交

git reset --soft HEAD^

R7：查看提交记录

git log

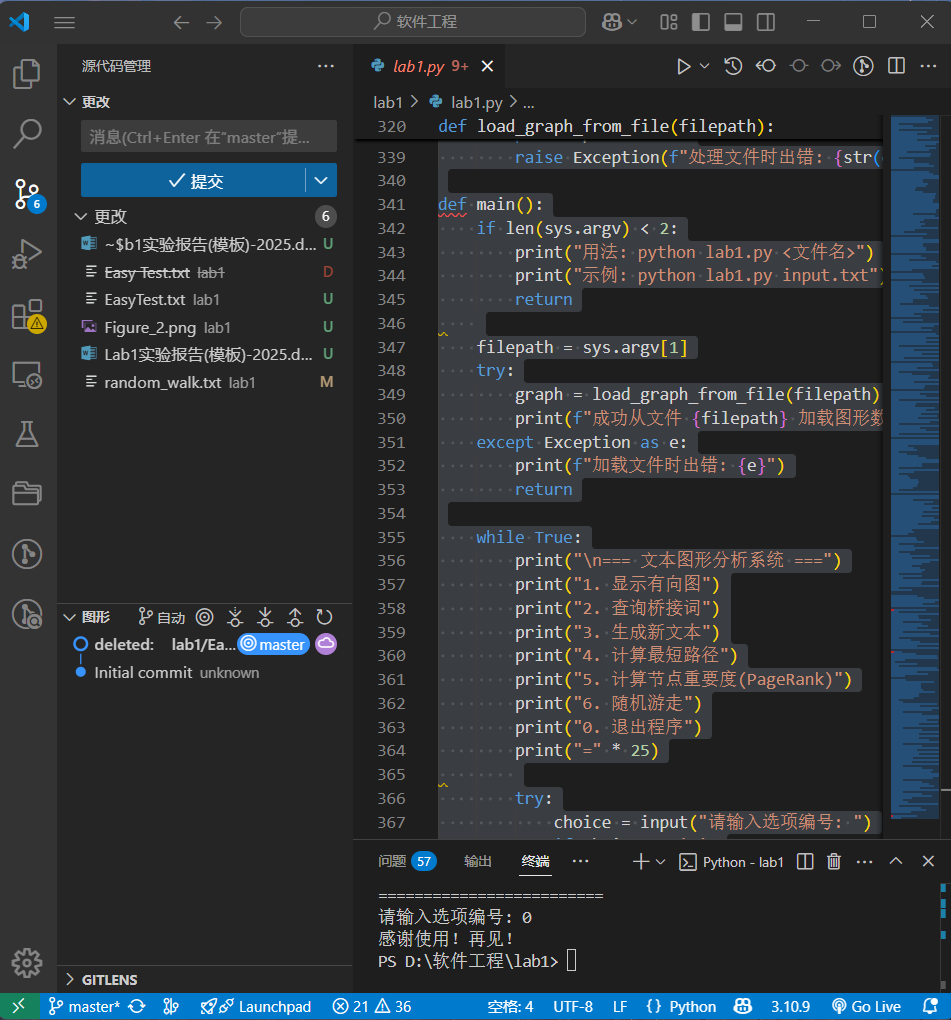
R8：创建远程仓库并关联本地仓库

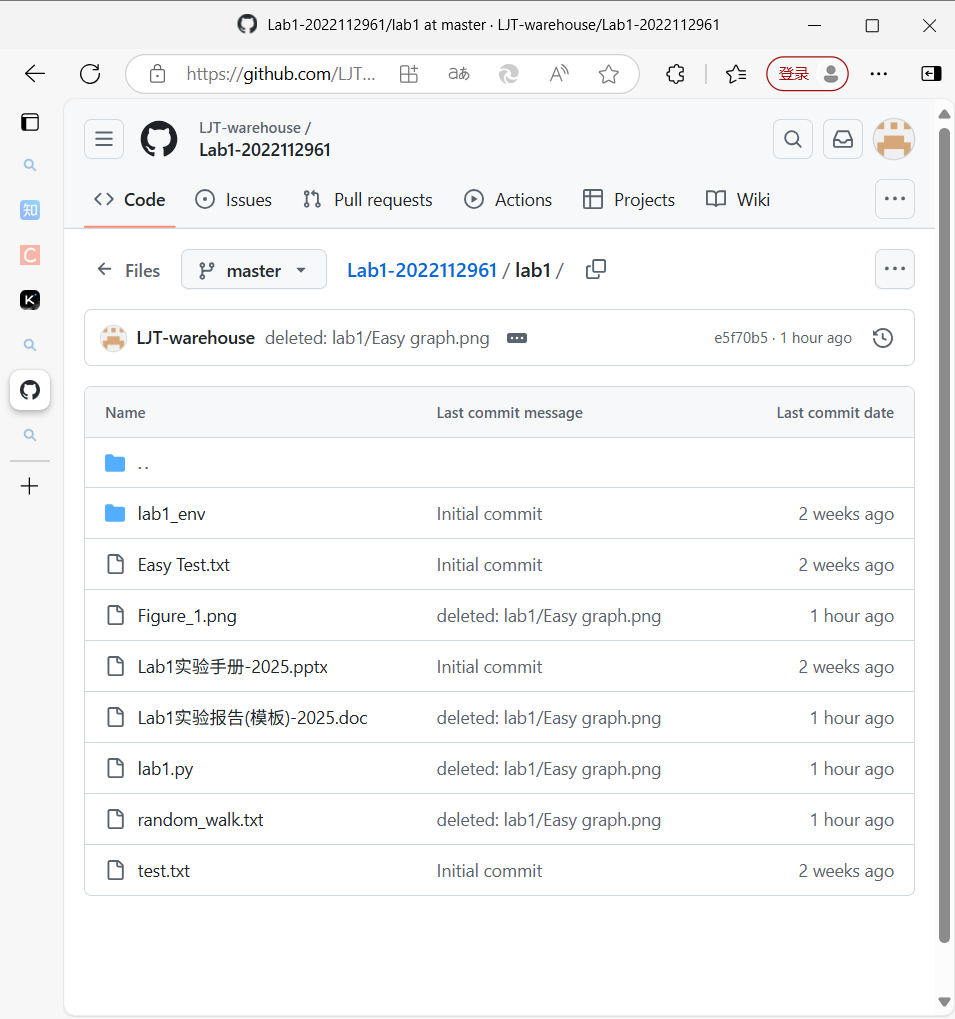
在 GitHub 上创建仓库后，复制仓库 URL

git remote add origin <\_url>

R9：推送本地仓库到 GitHub

git push -u origin master





## **实验场景(2)：分支管理**

**R1：查看和切换分支**

**git branch**

**git checkout master**

**R2：创建新分支 B1 和 B2**

**git branch B1**

**git branch B2**

**R3：在 B2 基础上创建 C4 分支**

**git checkout B2**

**git checkout -b C4**

**R4：在 C4 上修改文件并提交**

**使用文本编辑器修改文件**

**git add .**

**git commit -m "Changes on C4"**

**R5：在 B1 上修改相同文件并提交**

**git checkout B1**

**使用文本编辑器进行不同修改**

**git add .**

**git commit -m "Changes on B1"**

**R6：合并 C4 到 B1 并解决冲突**

**git merge C4**

**如果有冲突，手动解决后**

**git add .**

**git commit -m "Merged C4 into B1"**

**R7：在 B2 上修改文件并提交**

**git checkout B2**

**使用文本编辑器修改文件**

**git add .**

**git commit -m "Changes on B2"**

**R8：查看合并状态**

**git branch --merged**

**git branch --no-merged**

**R9：删除已合并分支并合并未合并分支到新分支（学号命名）**

**git branch -d C4**

**git checkout -b < \_id>**

**git merge B2**

**R10：推送新分支到 GitHub**

**git push origin < \_id>**

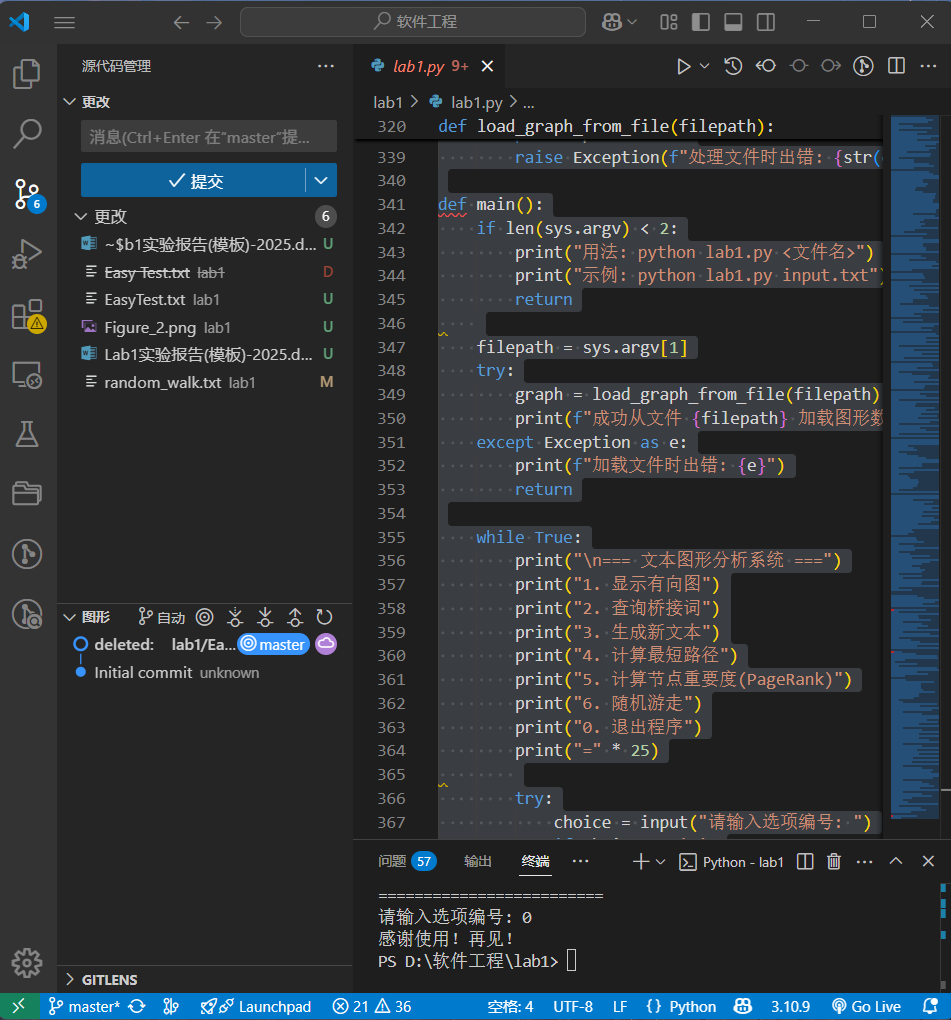
**R11：查看版本变迁树**

**git log --graph --oneline --all**

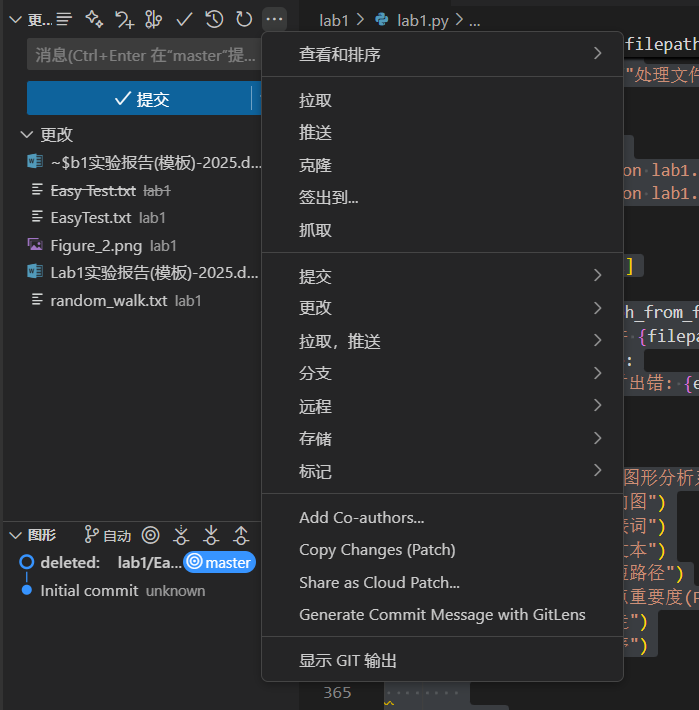
**R12：在 GitHub 上查看仓库状态**

**打开浏览器访问 GitHub 仓库页面**

# **在IDE中使用Git Plugin**



先提交至暂存区，然后点下图中的提交即可



# **小结**

通过本次实验，深刻体会到大模型辅助编程的优势与劣势：

▪ 优势：大模型能够快速提供思路和代码示例，帮助理解复杂概念和算法实现，加快开发进度。例如在设计有向图数据结构和 PageRank 算法时，参考大模型给出的思路，能够更高效地构思解决方案。

▪ 劣势：大模型生成的代码可能不够精准，需要进一步分析和修改。同时，过度依赖大模型可能会削弱自主思考和深入理解问题的能力。在实验过程中，部分代码经大模型生成后，仍需人工调整以适应实际需求和优化性能。

本次实验综合运用了面向对象编程、图算法、Git 版本控制等知识，提升了编程实践和

团队协作能力。通过完成各项功能需求，深入理解了文本处理、图操作和算法实现等关键技术点，为后续软件开发实践奠定了坚实基础。