

**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** |
| 任柏润 | 2022110620 | 18500822411 |

**目 录**

[1 实验要求 1](#_Toc198070204)

[2 待求解问题描述 1](#_Toc198070205)

[3 算法与数据结构设计 1](#_Toc198070206)

[3.1 设计思路与算法流程图 1](#_Toc198070207)

[功能1：从文本生成有向图 1](#_Toc198070208)

[3.2 数据结构设计 3](#_Toc198070209)

[3.3 算法时间复杂度分析 4](#_Toc198070210)

[功能1：从文本生成有向图 4](#_Toc198070211)

[复杂度：时间复杂度为O(N+M)((N为文本字符数，M为单词总数) 4](#_Toc198070212)

[3.4 算法代码的生成 5](#_Toc198070213)

[4 实验与测试 5](#_Toc198070214)

[4.1 读取文本文件并展示有向图 5](#_Toc198070215)

[4.2 查询桥接词 6](#_Toc198070216)

[4.3 根据桥接词生成新文本 8](#_Toc198070217)

[4.4 计算最短路径 10](#_Toc198070218)

[4.5 计算PageRank值 13](#_Toc198070219)

[4.6 随机游走 15](#_Toc198070221)

[5 编程语言与开发环境 16](#_Toc198070222)

[6 Git操作过程 16](#_Toc198070223)

[6.1 实验场景(1)：仓库创建与提交 16](#_Toc198070224)

[6.2 实验场景(2)：分支管理 18](#_Toc198070225)

[7 在IDE中使用Git Plugin 18](#_Toc198070226)

[8 小结 18](#_Toc198070227)

[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

开发一个程序，实现从文本文件中读取数据并根据要求生成图结构，输出该图结构，并在其上进行一系列计算操作，实时展示各操作的结果。开发的程序可以是命令行方式运行，也可以用图形化用户界面GUI的方式运行。无论何种方式，均应覆盖后续所有功能需求。

# 待求解问题描述

输入一个文本文件，其中包含用英文书写的文本数据；文本分为多行，程序默认将换行/回车符当作空格；文本中的任何标点符号，也应当作空格处理；文本中的非字母(A-Z和a-z之外)字符被忽略。

需求功能一：程序读入文本数据，进行分析，将其转化为有向图，有向图的节点为文本中包含的某个单词，节点A->B的权重w=文本中A和B相邻出现的次数。

需求功能二：展示有向图，通过自定义的格式在CLI（命令行界面）上进行展示

需求功能三：在生成有向图之后，用户输入任意两个英文单词word1、word2，程序从图中查询它们的“桥接词”，需要对不存在桥接词，存在一个或多个桥接词的情况分别处理。

需求功能四：输入一行新文本，程序根据之前输入文件生成的图，计算该新文本中两两相邻的单词的桥接词，将桥接词插入新文本的两个单词之间，输出到屏幕上展示；需要对无桥接词情况，多桥接词情况分别处理。

需求功能五：计算两个单词之间的最短路径

需求功能六：计算有向图每个节点的PageRank

需求功能七：实现随机游走，程序随机的从图中选择一个节点，以此为起点沿出边进行随机遍历，记录经过的所有节点和边，直到出现第一条重复的边为止，或者进入的某个节点不存在出边为止。

# 算法与数据结构设计

## 设计思路与算法流程图

### 功能1：从文本生成有向图

文本预处理：

过滤非字母字符：使用正则表达式 [^a-zA-Z] 替换所有非字母字符为空格。

统一小写处理：将所有单词转换为小写，确保不区分大小写。

分词处理：按空格分割文本为单词列表，过滤空字符串。

构建邻接表：

遍历相邻词对：对单词列表逐对遍历（当前词与下一词）。

统计边权重：用嵌套字典 defaultdict 记录每条边 A→B 的出现次数作为权重。

功能2：展示有向图

命令行展示（CLI）：

边收集与排序：收集所有边并按字母顺序排序，确保输出一致性。

多列格式化：根据终端宽度动态计算每行可显示的边数，优化可读性。

统计摘要：计算总节点数、总边数和最大出度。

图形文件保存：

DOT语言生成：使用 graphviz.Digraph 定义节点和带权边。

依赖处理：自动检测 graphviz 库和系统环境，提供友好错误提示。

功能3：查询桥接词

输入验证：将输入单词转为小写，检查是否存在于图中。

桥接词搜索：

候选词筛选：遍历 word1 的所有出边节点 word3。

反向验证：检查 word3 是否指向 word2。

结果格式化：

处理单数、复数及连接的语法逻辑。

功能需求4：生成新文本

输入处理：

对用户输入的文本进行相同的预处理（过滤、分词、小写化）。

桥接词插入：

逐对分析：遍历输入文本的相邻词对。

桥接词查询：对每对词调用 queryBridgeWords。

随机选择：若存在多个桥接词，使用 random.choice 随机插入一个。

输出重构：

保留原始输入单词的大小写格式，仅插入小写的桥接词。

功能需求5：计算最短路径

Dijkstra算法实现：

优先级队列：使用 heapq 实现最小堆，优化节点选择效率。

距离更新：按边权重累加路径长度，权重为共现次数

路径回溯：通过前驱节点字典反向构建路径。

特殊处理：

节点存在性检查：验证起始词和终点词是否在图中。

不可达处理：若终点词距离为无穷大，返回错误信息。

功能需求6：计算PageRank

迭代公式实现：

带权公式：PR(u) = (1-d)/N + d \* Σ(PR(v) \* w(v→u) / W(v))，其中 W(v)为节点v的出边总权重。

悬挂节点处理：将无出边节点的PR值均分给所有节点。

收敛优化：

阈值判断：当两次迭代的PR值总变化小于  时提前终止。

归一化：最终结果按总和为1进行归一化。

功能需求7：随机游走

遍历逻辑：

起点选择：从图中随机选择一个初始节点。

权重随机游走：使用 random.choices(weights=边权重) 选择下一节点。

终止条件：

遇到重复边（记录边的元组 (A, B)）。

进入无出边节点。

用户手动中断（通过非阻塞输入检测）。

统一响应：回车键触发终止标志。

路径保存：

用空格分隔节点，保存为纯文本文件。

## 数据结构设计

功能需求1：从文本生成有向图

邻接表（Adjacency List）：

类型：defaultdict[str, dict[str, int]]

定义：外层字典的键为源节点（单词），值为内层字典；内层字典的键为目标节点（单词），值为边权重（相邻共现次数）。

用途：存储图的边和权重，高效支持按源节点查询所有出边及权重。

功能需求2：展示有向图

边列表（Edge List）：

类型：list[tuple[str, str, int]]

定义：列表元素为 (源节点, 目标节点, 权重) 的三元组。

用途：收集所有边并按字母排序后格式化输出。

Graphviz图对象：

类型：graphviz.Digraph

定义：外部库 graphviz 的图对象，通过节点和边的添加生成可视化图形。

用途：生成并保存图的图形文件。

功能需求3：查询桥接词

桥接词候选列表：

类型：list[str]

定义：存储满足 word1→word3→word2 条件的所有中间词。

用途：收集并返回所有可能的桥接词。

功能需求4：生成新文本

输入文本单词列表：

类型：list[str]

定义：用户输入文本预处理后的单词序列（保留原始大小写）。

用途：遍历相邻词对并插入桥接词。

扩展后的文本列表：

类型：list[str]

定义：动态修改的单词列表，包含插入的桥接词。

用途：最终拼接为生成的新文本。

功能需求5：计算最短路径

距离字典（Distance Map）：

类型：dict[str, float]

定义：键为节点，值为从起点到该节点的当前最短距离。

用途：Dijkstra 算法中记录最短路径长度。

前驱字典（Predecessor Map）：

类型：dict[str, str | None]

定义：键为节点，值为到达该节点的前驱节点。

用途：回溯生成完整路径。

优先队列（Priority Queue）：

类型：list[tuple[float, str]]

定义：堆结构元素为 (距离, 节点)，按距离排序。

用途：高效获取当前未处理的最小距离节点。

功能需求6：计算PageRank

PageRank值字典：

类型：dict[str, float]

定义：键为节点，值为归一化后的PageRank值。

用途：存储迭代计算的节点重要度。

节点出边总权重字典：

类型：dict[str, int]

定义：键为节点，值为该节点所有出边的权重之和。

用途：计算节点贡献时的分母。

功能需求7：随机游走

路径列表（Walk Path）：

类型：list[str]

定义：按遍历顺序记录的节点序列。

用途：保存游走结果并输出到文件。

已访问边集合（Visited Edges）：

类型：set[tuple[str, str]]

定义：元素为 (源节点, 目标节点) 的元组。

用途：检测重复边以终止遍历。

## 算法时间复杂度分析

### 功能1：从文本生成有向图

### 复杂度：时间复杂度为O(N+M)((N为文本字符数，M为单词总数)

功能2：展示有向图

复杂度：时间复杂度为O(ElogE)（E为边数）

功能3：查询桥接词

复杂度：O(K)(K为 word1 的出边节点数）。

功能4：生成新文本

复杂度：O(L+P\*K), 其中L为输入文本字符数。其中P为输入词对数，K为平均桥接词查询时间。

功能5：计算最短路径

复杂度：O((E+N)logN)，其中 N为节点数，E为边数（优先队列操作）。

功能6：计算PageRank

复杂度：O(I\*(N+E))，其中I为迭代次数。

功能需求7：随机游走

复杂度：O(D)，其中D为路径长度

## 算法代码的生成

七个功能的提示词分别为：读入文本并生成有向图，CLI展示有向图，查询桥接词，根据桥接词生成中间文本，计算两个单词之间的最短路径，计算每个单词的PageRank，随机遍历。

# 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）：

'a' -> 'detailed' [w=1] 'analyzed' -> 'it' [w=1]

'analyzed' -> 'the' [w=1] 'and' -> 'shared' [w=1]

'but' -> 'the' [w=1] 'carefully' -> 'analyzed' [w=1]

'data' -> 'so' [w=1] 'data' -> 'wrote' [w=1]

'detailed' -> 'report' [w=1] 'it' -> 'again' [w=1]

'more' -> 'data' [w=1] 'report' -> 'and' [w=1]

'report' -> 'with' [w=1] 'requested' -> 'more' [w=1]

'scientist' -> 'analyzed' [w=1] 'scientist' -> 'carefully' [w=1]

'shared' -> 'the' [w=1] 'so' -> 'the' [w=1]

'team' -> 'but' [w=1] 'team' -> 'requested' [w=1]

'the' -> 'data' [w=1] 'the' -> 'report' [w=1]

'the' -> 'scientist' [w=2] 'the' -> 'team' [w=2]

'with' -> 'the' [w=1] 'wrote' -> 'a' [w=1]

程序实际生成的图：



二者是否一致：一致

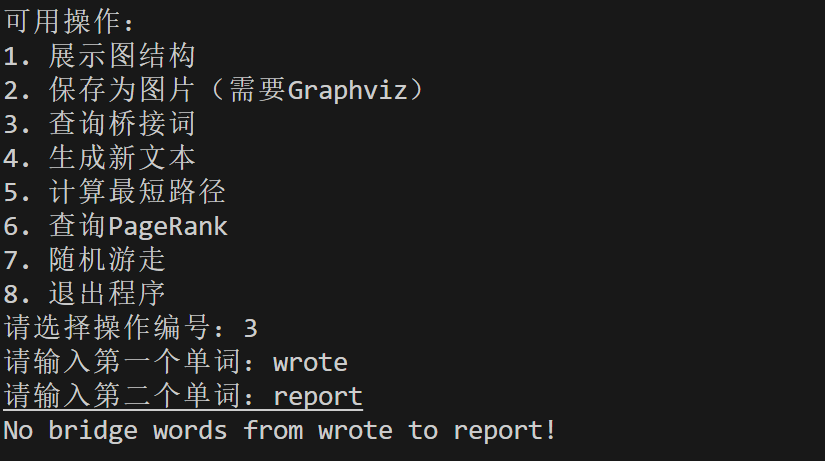
给出实际运行得到结果的界面截图。

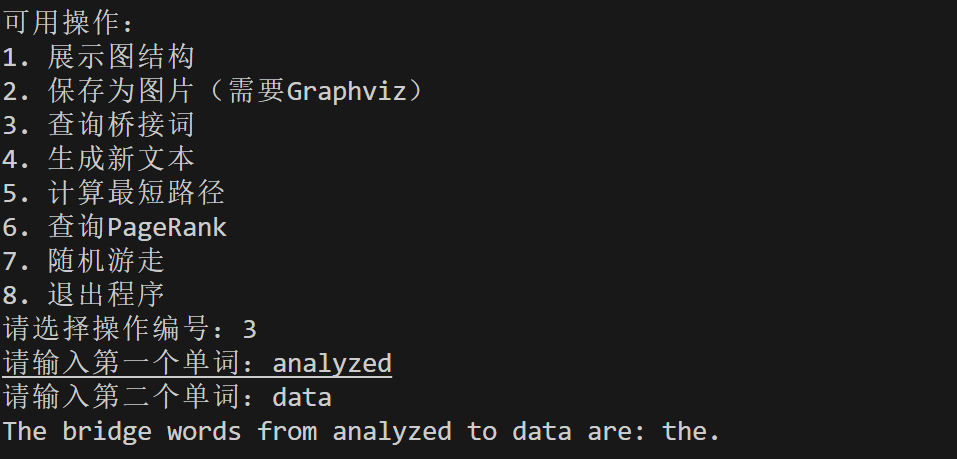


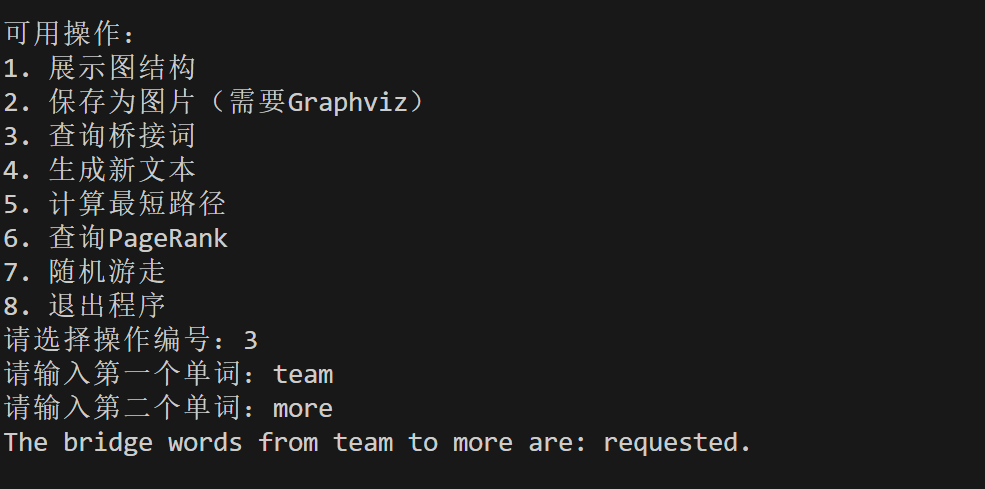
## 查询桥接词

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 输入（2个单词） | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | wrote,report | No bridge words from wrote to report! | No bridge words from wrote to report! | 正确 |
| 2 | analyzed,data | The bridge words from analyzed to data are: the. | The bridge words from analyzed to data are: the. | 正确 |
| 3 | team,more | The bridge words from team to more are: requested. | The bridge words from team to more are: requested. | 正确 |

给出实际运行得到结果的界面截图。



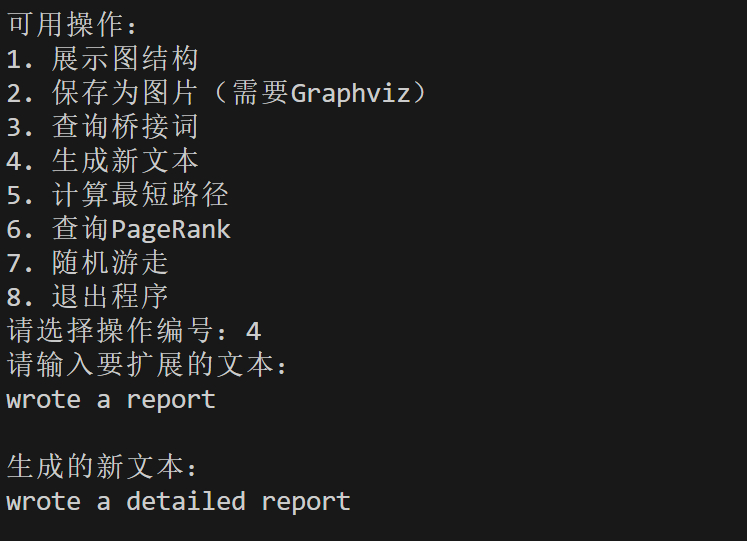


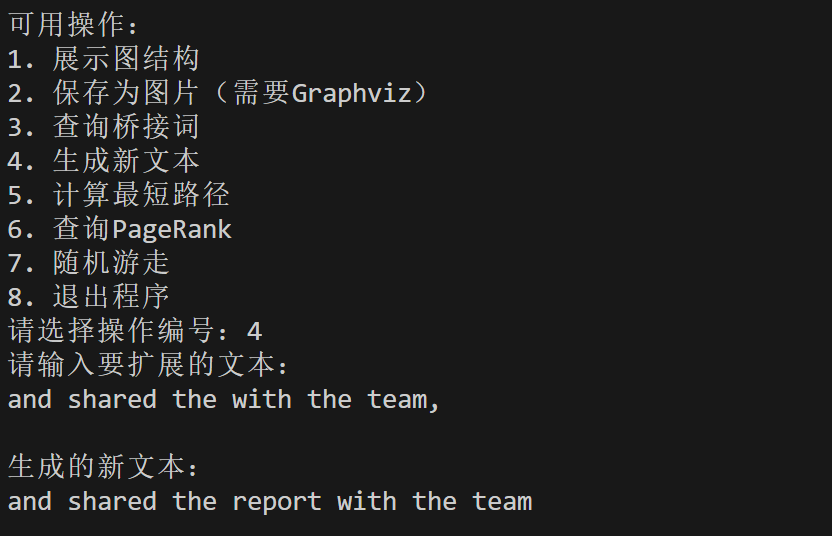


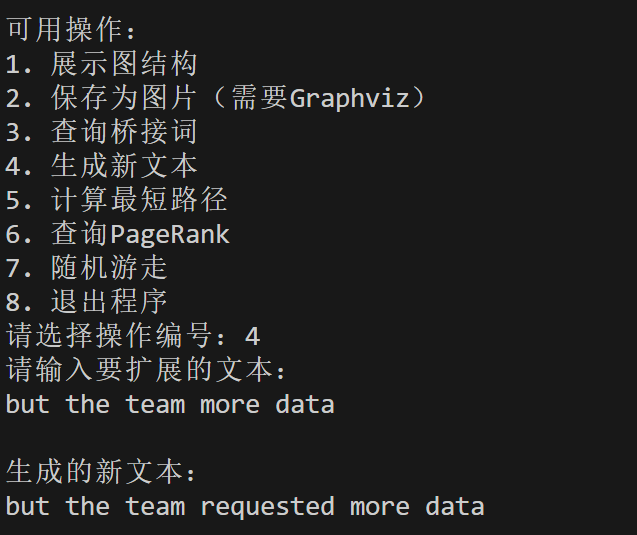
## 根据桥接词生成新文本

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 输入（一行文本） | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | wrote a report | wrote a detailed report | wrote a detailed report | 正确 |
| 2 | and shared the with the team, | and shared the report with the team | and shared the report with the team | 正确 |
| 3 | but the team more data | but the team requested more data | but the team requested more data | 正确 |

给出实际运行得到结果的界面截图。



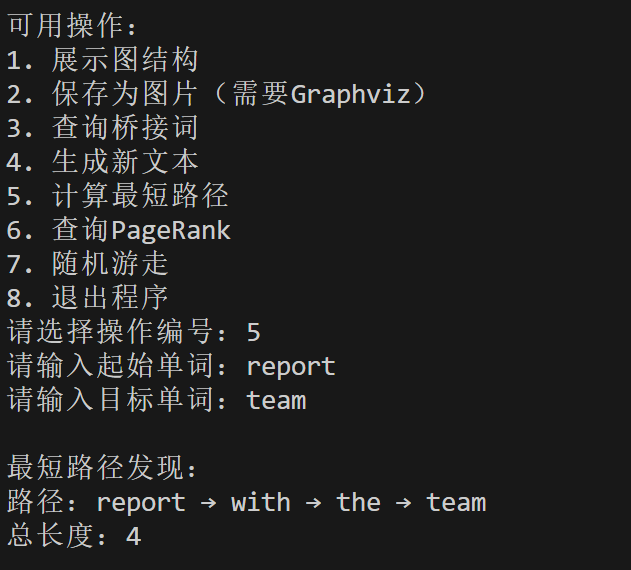


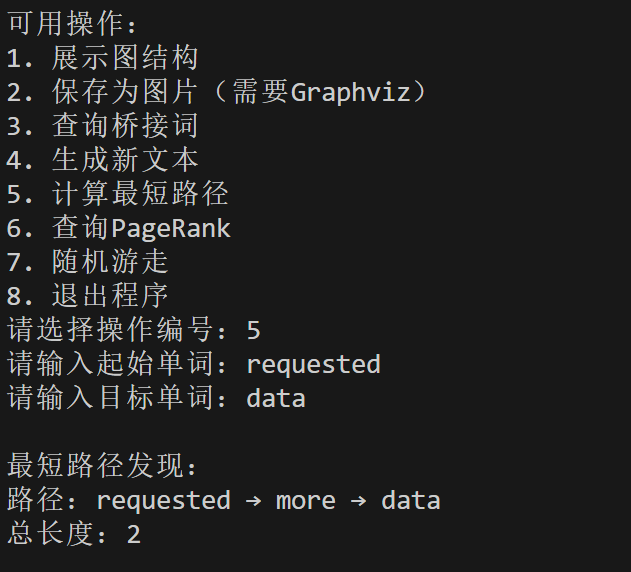


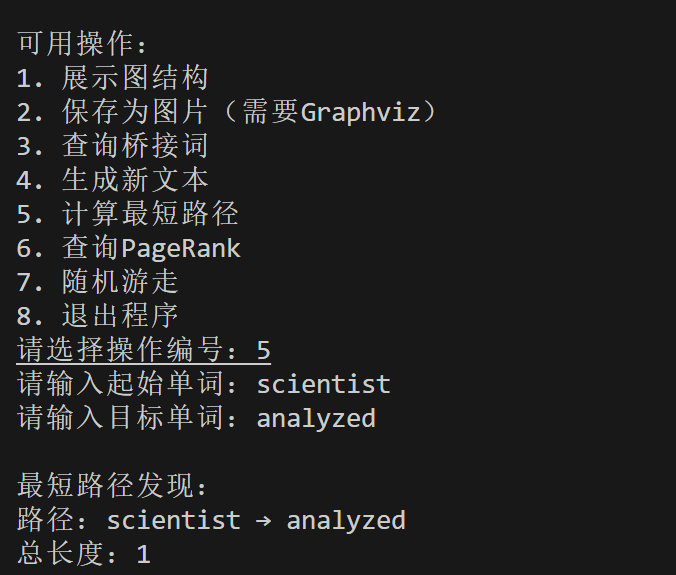
## 计算最短路径

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | report team | report → with → the → team | report → with → the → team | 正确 |
| 2 | requested data | requested → more → data | requested → more → data | 正确 |
| 3 | scientist analyzed | scientist → analyzed | scientis→ analyzed | 正确 |

给出实际运行得到结果的界面截图。



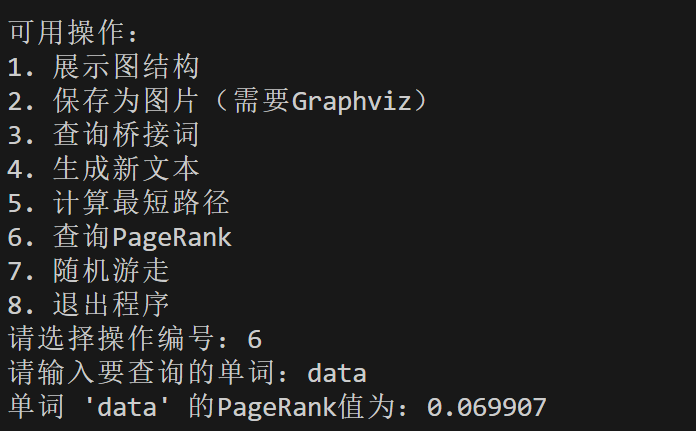


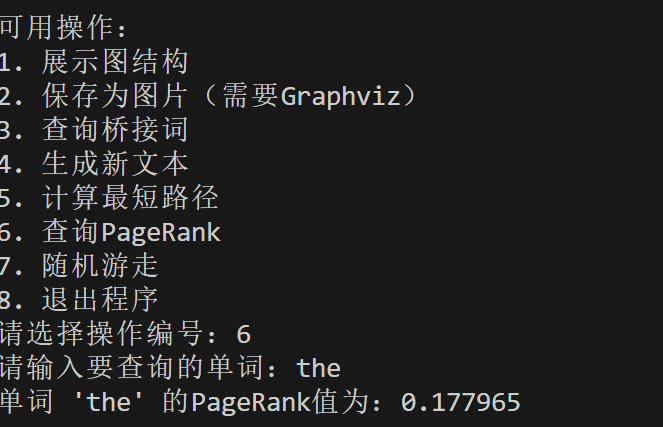


## 计算PageRank值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 单词 | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | report | 0.075785 | 0.075785 | 正确 |
| 2 | data | 0.069907 | 0.069907 | 正确 |
| 3 | the | 0.177965 | 0.177965 | 正确 |

## 



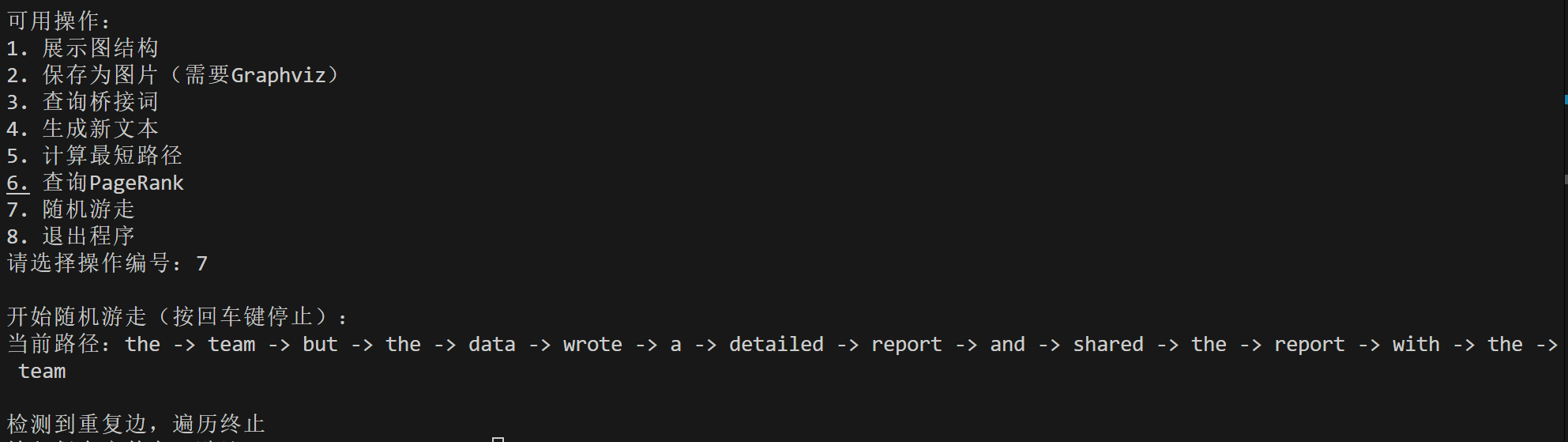


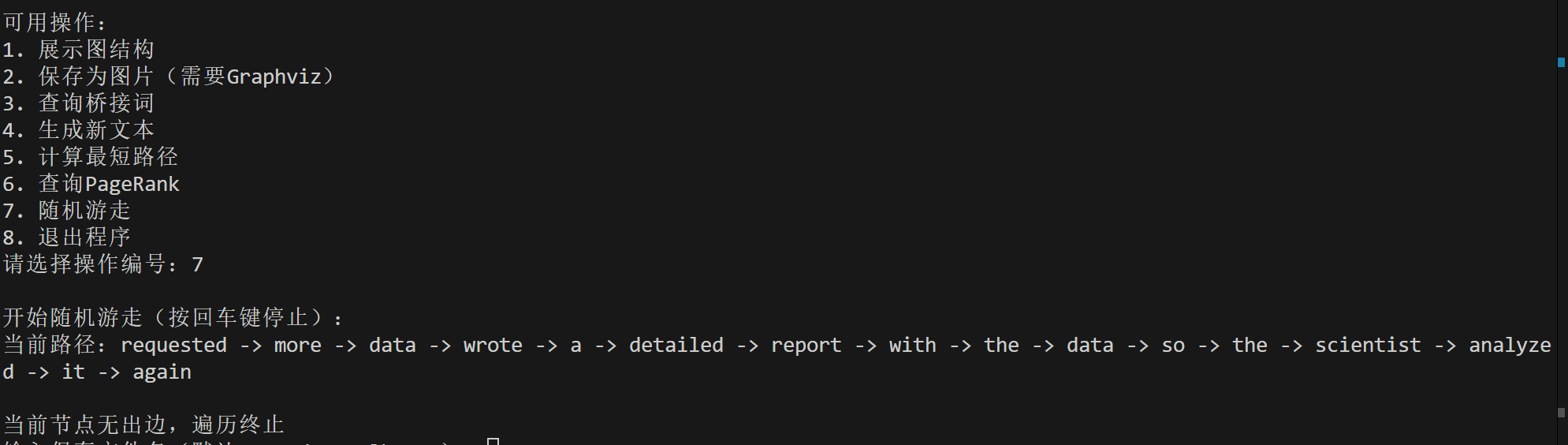
## 随机游走

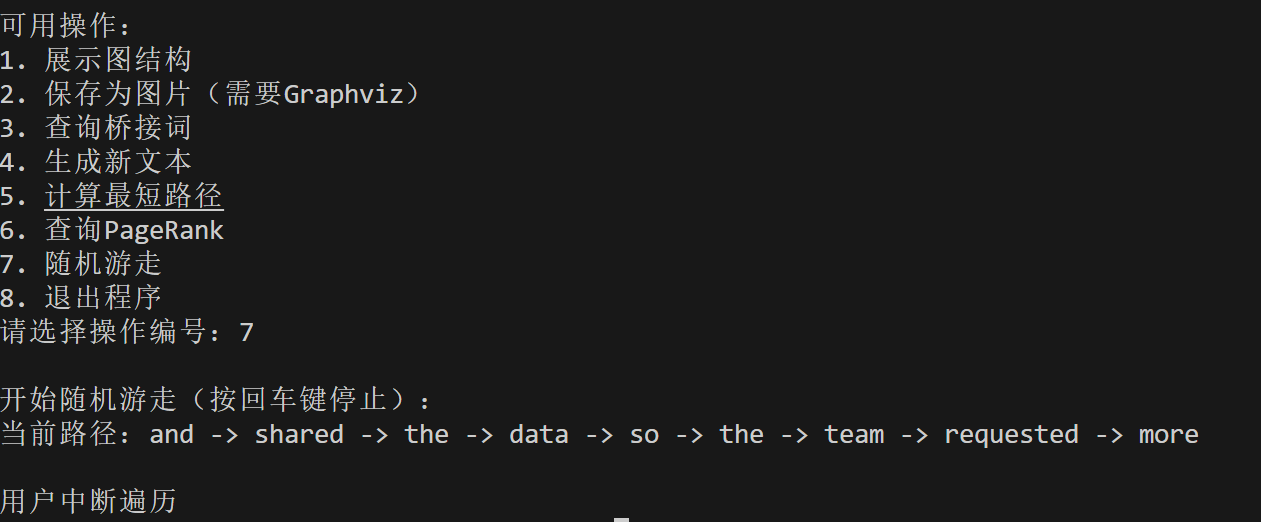
该功能无输入，让你的程序执行多次，分别记录结果。

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

环境：Vscode 1.92.2

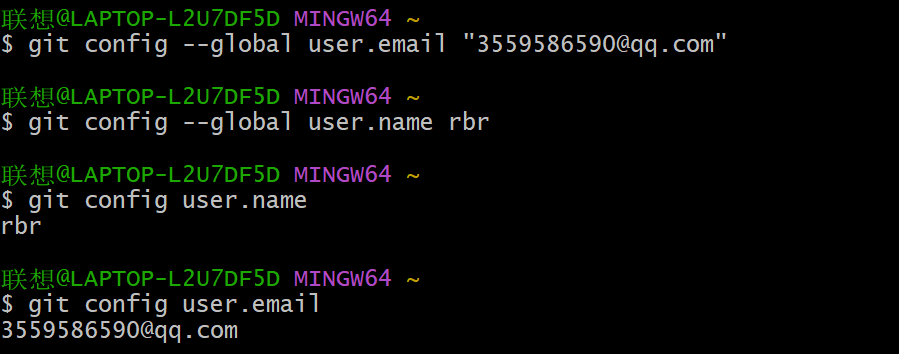
语言：python 3.12.5

大模型：deepseek-R1

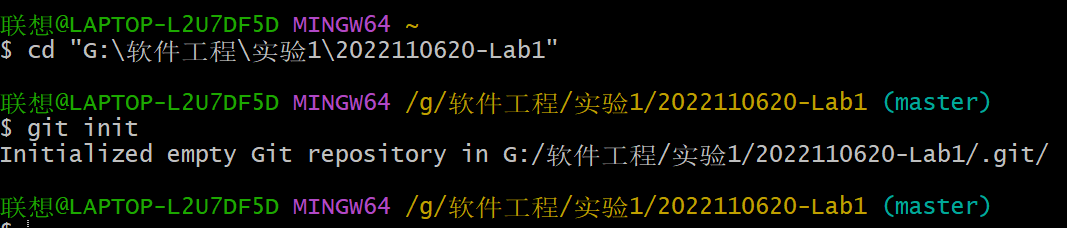
# Git操作过程

## 实验场景(1)：仓库创建与提交

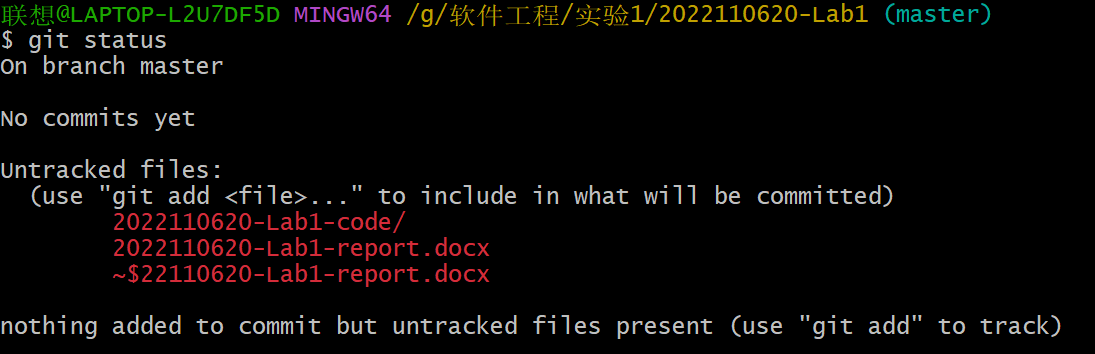
Git初始化：



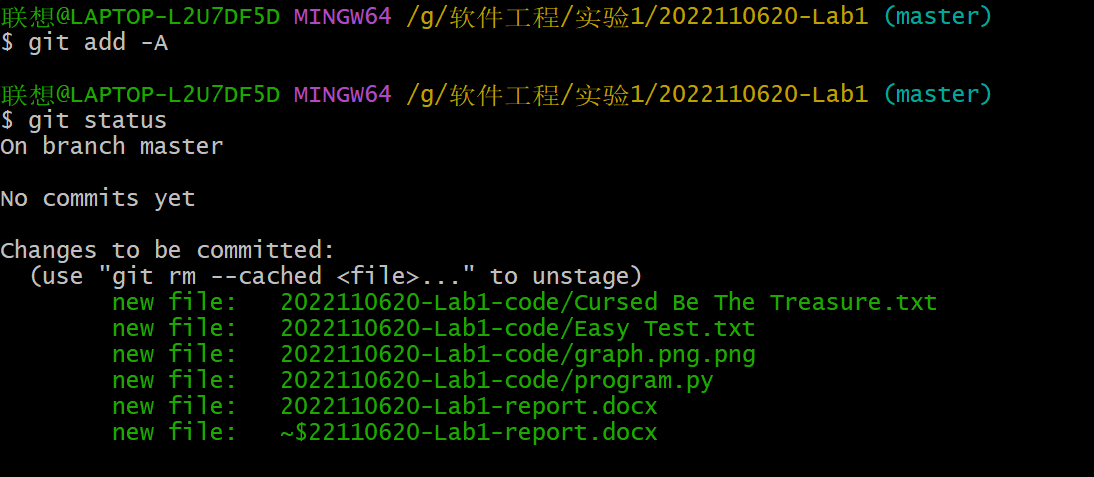
新建本地git仓库：



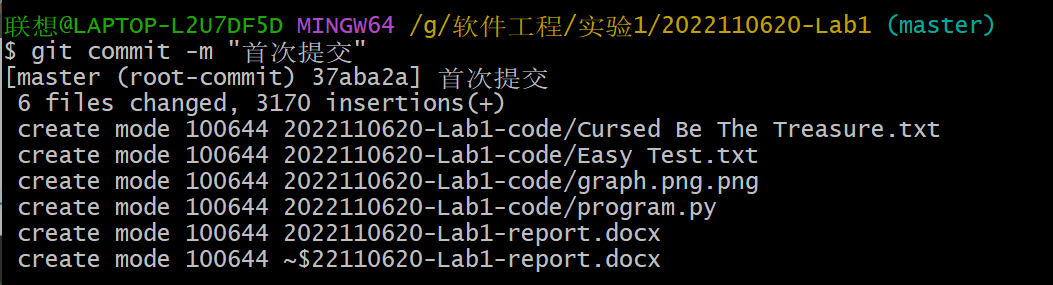
查询git仓库状态：



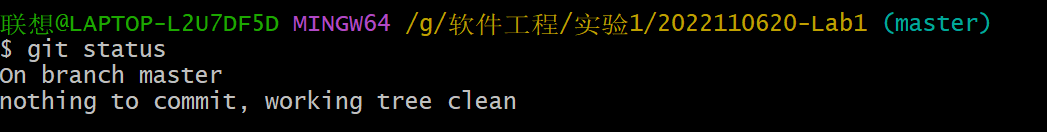
使用git add -A命令跟踪目录下的报告和代码，并加入暂存区



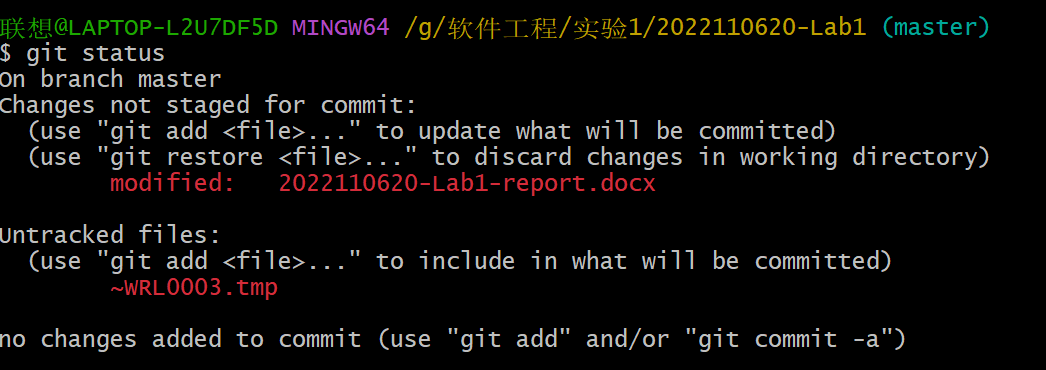
将目录下的所有文件首次提交

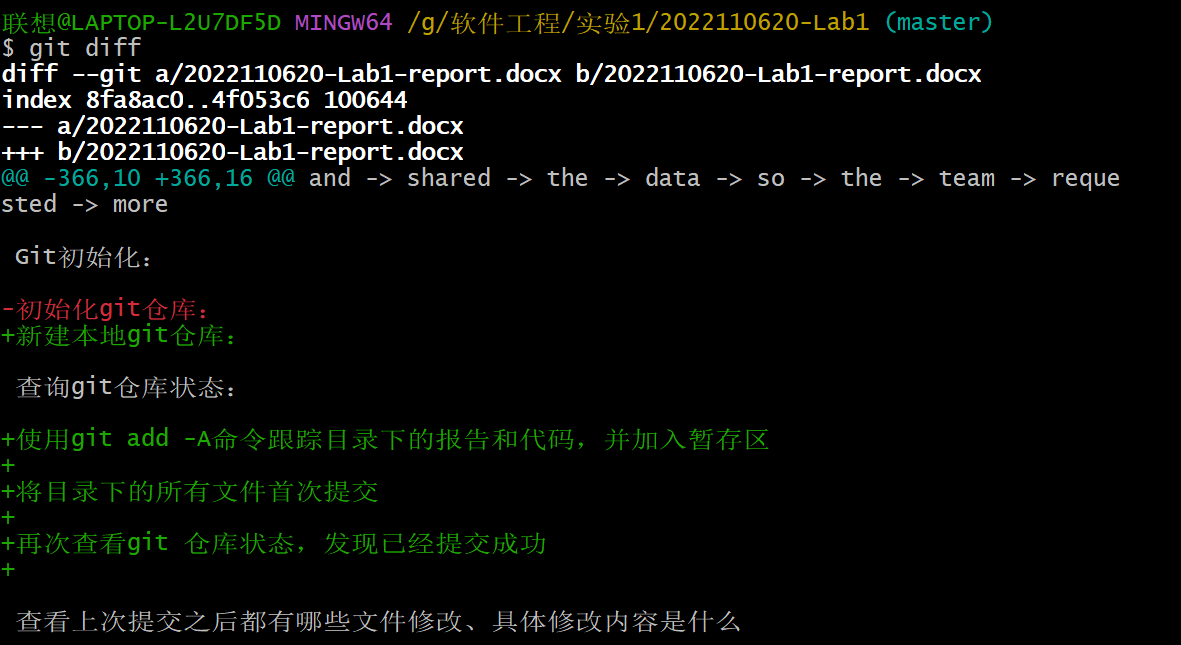


再次查看git仓库状态，发现已经提交成功

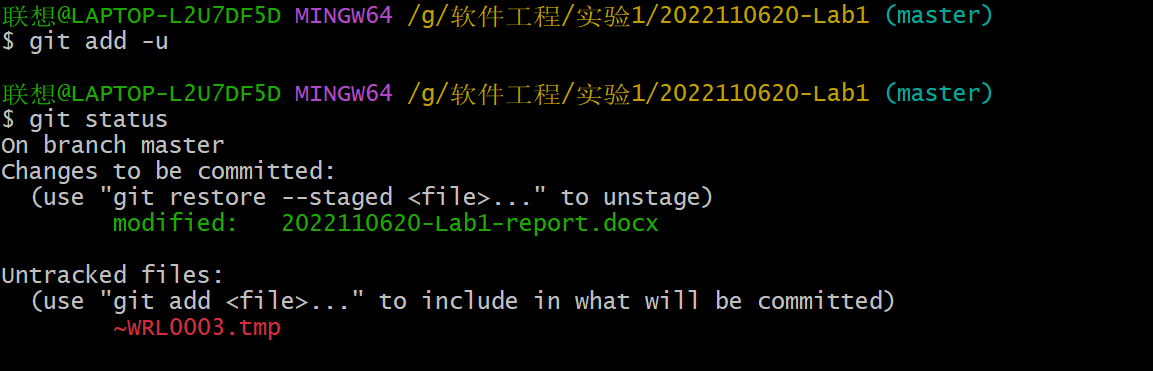


对实验报告进行修改，再次查看git仓库状态：

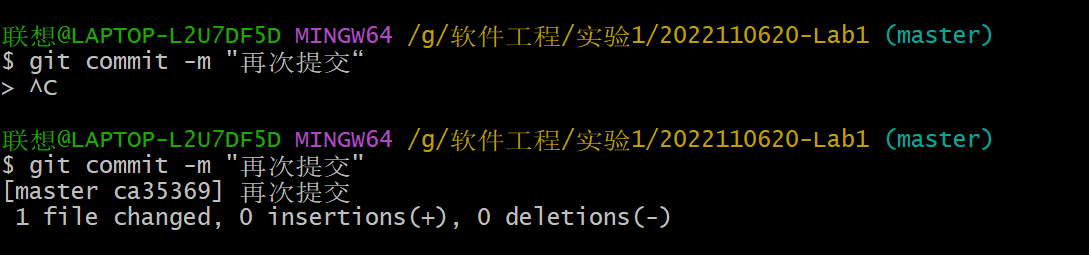
查看上次提交之后都有哪些文件修改、具体修改内容是什么：

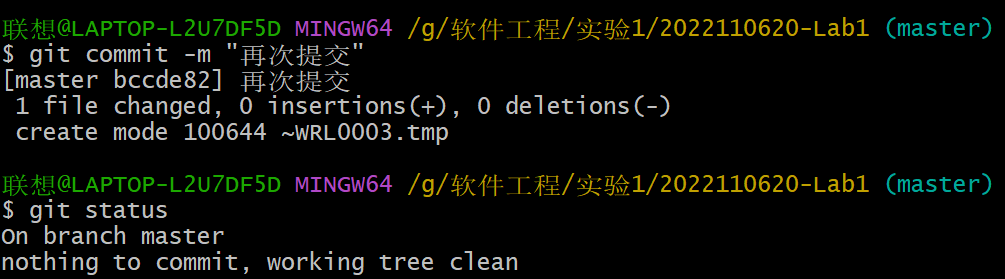


暂存已经修改的文件：

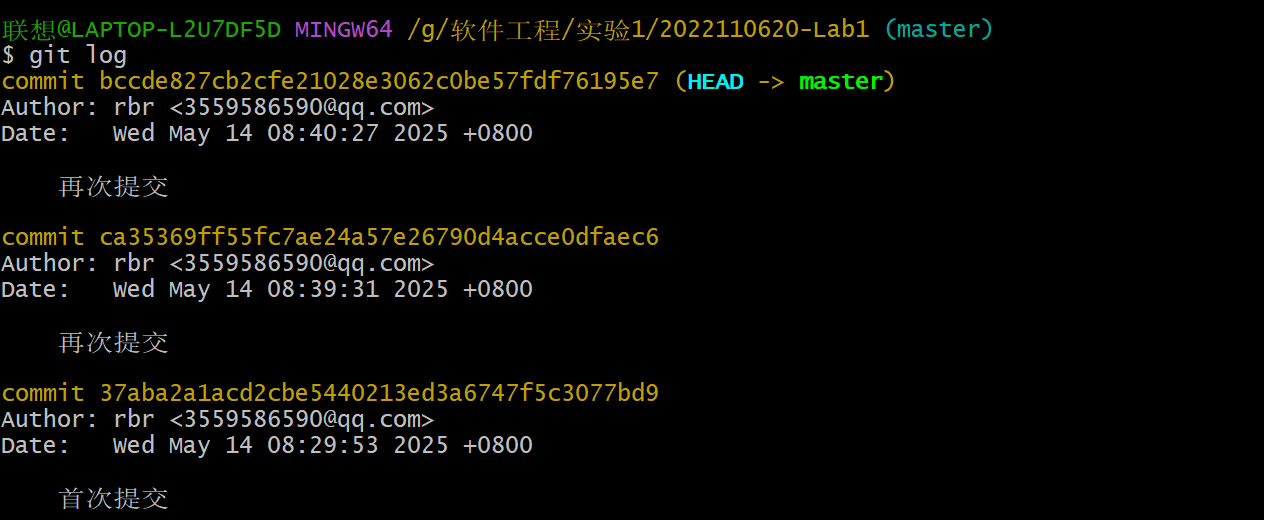


再次提交：

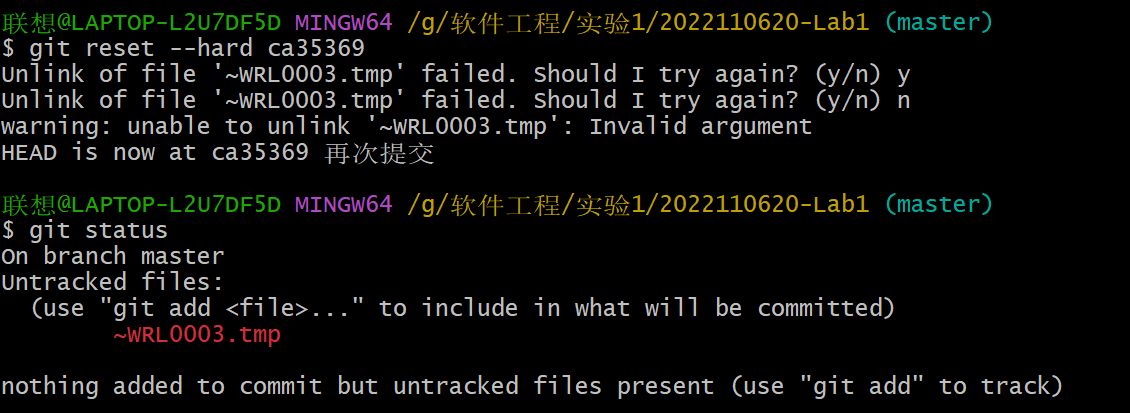




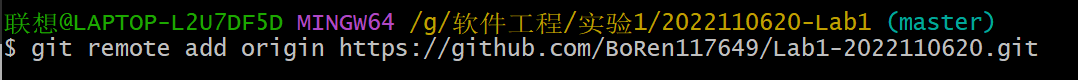
查询提交记录：



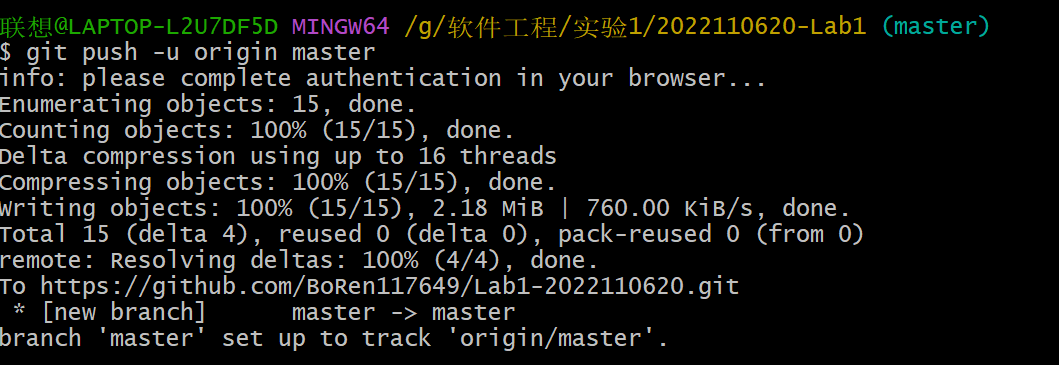
撤销最后一次提交记录：

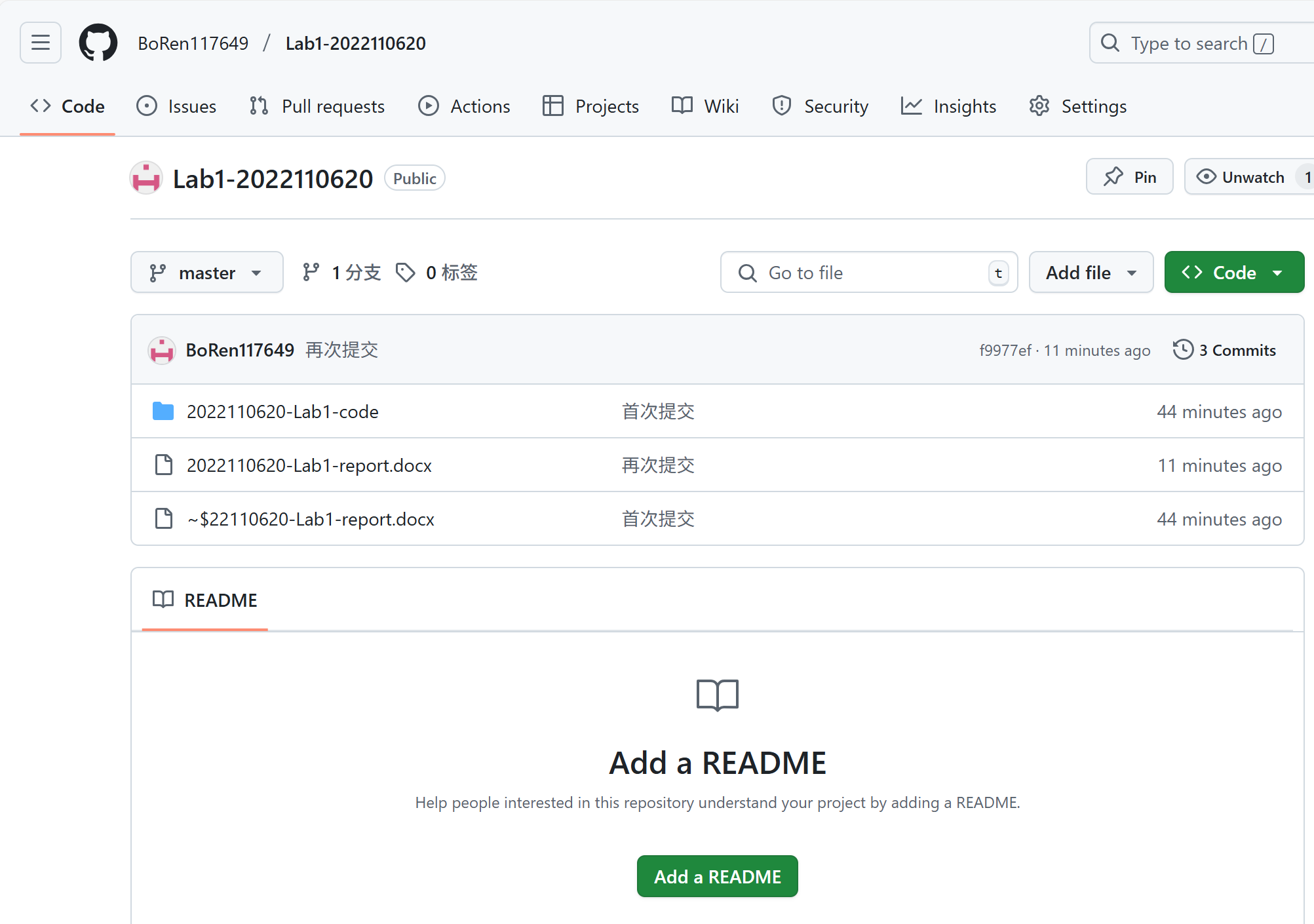


建立远程仓库并与github连接：



推送本地仓库内容进入github：

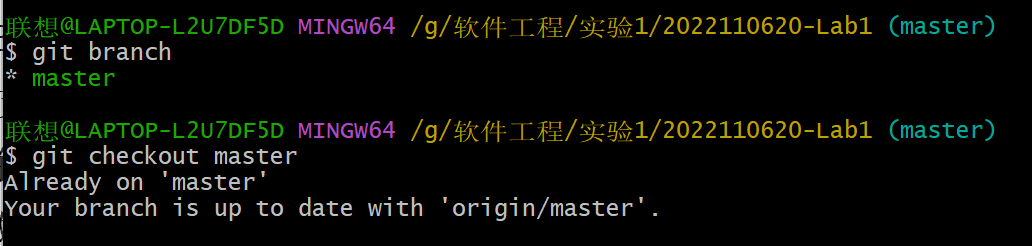




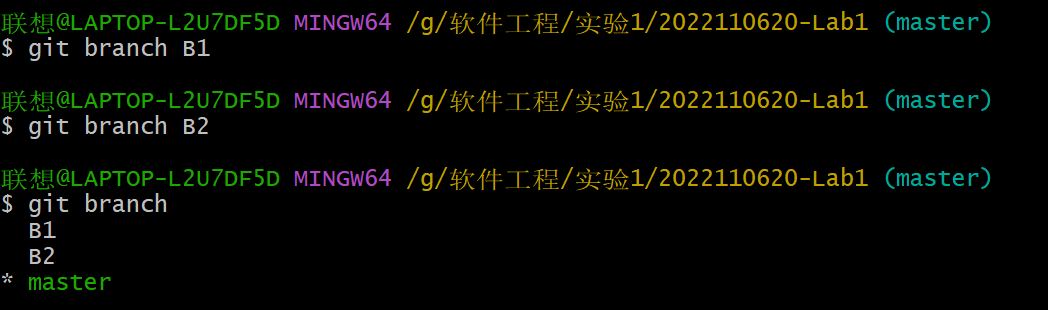
提交成功！

## 实验场景(2)：分支管理

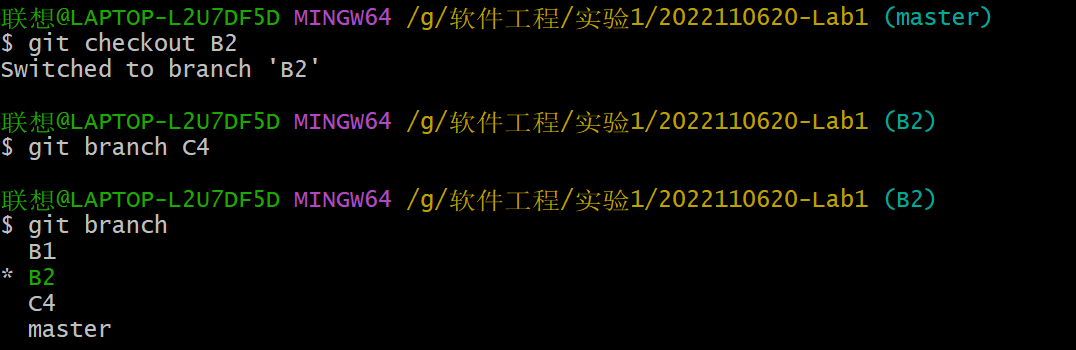
获得全部分支，切换到分支master：



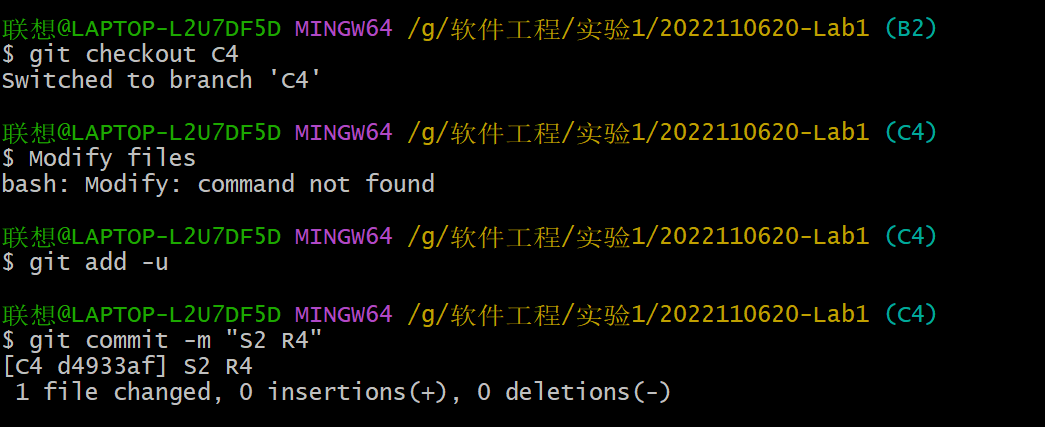
建立分支B1，B2



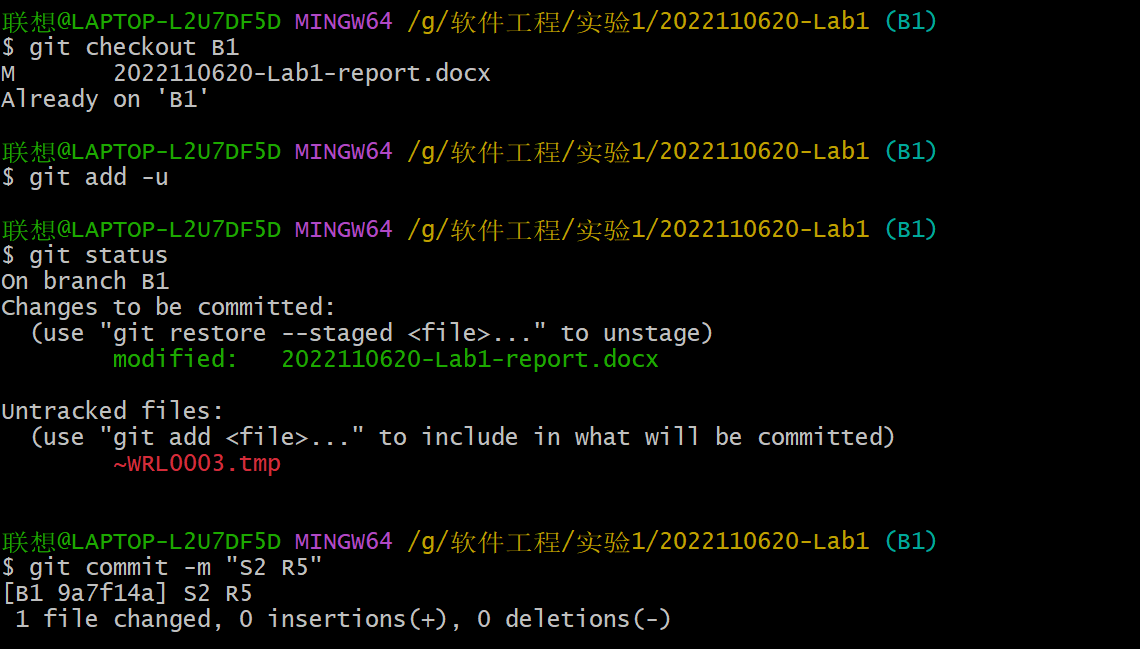
在B2基础上创建新分支C4



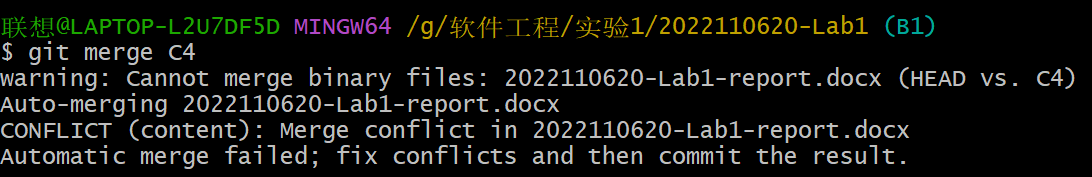
在C4上，对2个文件进行修改并提交



在B1分支上对同样的2个文件做不同修改并提交；



合并时出现冲突：



# 在IDE中使用Git Plugin

**采用屏幕截图和文字说明的方式，给出在IDE中使用Git plugin向本地仓库和Github提交Lab1的命令或界面。**

# 小结

优势（大模型辅助编程带来的提升）：

(1) 开发效率显著提高

代码生成：大模型能快速生成基础代码（如函数、类、API 调用），减少手动编写时间。

自动补全：智能提示减少打字量，如 Copilot 能预测整行或整段代码。

减少重复劳动：模板代码（如 CRUD 接口、SQL 查询）可一键生成。

(2) 降低学习成本

快速学习新技术：遇到不熟悉的框架/语言时，可直接让 AI 解释并提供示例代码。

减少文档查阅时间：不用频繁搜索 Stack Overflow 或官方文档，AI 能直接给出答案。

(3) 调试与优化更高效

错误修复：粘贴报错信息，AI 能快速定位问题并提供修复建议。

代码优化：AI 可建议性能优化方案（如算法改进、减少冗余计算）。

(4) 增强代码质量

代码审查：AI 能检测潜在 Bug（如内存泄漏、竞态条件）并建议改进。

最佳实践：自动推荐符合行业标准的写法（如 REST API 设计、安全编码）。

(5) 跨语言/技术栈支持

无缝切换语言：开发者可借助 AI 快速编写不熟悉的语言（如 Python → Rust）。

多技术栈集成：AI 能帮助整合不同工具（如 Docker + Kubernetes + AWS）。

2. 劣势（潜在问题与挑战）

(1) 代码质量依赖提示词（Prompt）

模糊需求导致低质量代码：如果问题描述不清，AI 可能生成错误或低效的实现。

过度依赖生成代码：开发者可能不深入理解生成的代码，导致维护困难。

(2) 安全性与合规风险

敏感信息泄露：向 AI 粘贴公司代码可能违反数据安全政策（如 Copilot 训练数据问题）。

依赖不可信代码：AI 可能推荐过时/不安全的依赖（如漏洞库版本）。

**。**