

**2025年春季学期  
计算学部《软件工程》课程**

**实验报告**

**Lab3 代码评审与单元测试**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| 常添 | 2022111699 | 13936408592@163.com |

**目 录**

[1 实验要求 1](#_Toc200271927)

[2 在IDE中配置代码审查与分析工具 1](#_Toc200271928)

[2.1 Flake8 1](#_Toc200271929)

[2.2 Bandit 1](#_Toc200271930)

[2.3 Converge 1](#_Toc200271931)

[2.4 Pytest 1](#_Toc200271932)

[3 Flake8所发现的代码问题清单及原因分析 2](#_Toc200271933)

[4 Bandit所发现的代码问题清单及原因分析 3](#_Toc200271934)

[5 针对Lab1的黑盒测试 4](#_Toc200271935)

[5.1 所选的被测函数及其需求规约 4](#_Toc200271936)

[5.2 等价类划分结果 4](#_Toc200271937)

[5.3 测试用例设计 5](#_Toc200271938)

[5.4 Pytest单元测试代码 5](#_Toc200271939)

[5.5 Pytest单元测试结果 6](#_Toc200271940)

[5.6 未通过测试的原因分析及代码修改 7](#_Toc200271941)

[5.7 Git操作记录 8](#_Toc200271942)

[6 针对Lab1的白盒测试 9](#_Toc200271943)

[6.1 所选的被测函数 9](#_Toc200271944)

[6.2 程序流程图 11](#_Toc200271945)

[6.3 控制流图 12](#_Toc200271946)

[6.4 圈复杂度计算与基本路径识别 12](#_Toc200271947)

[6.5 测试用例设计 13](#_Toc200271948)

[6.6 Pytest测试代码 13](#_Toc200271949)

[6.7 Pytest单元测试结果 15](#_Toc200271950)

[6.8 代码覆盖度分析 16](#_Toc200271951)

[6.9 未通过测试的原因分析及代码修改 16](#_Toc200271952)

[6.10 Git操作记录 17](#_Toc200271953)

[7 计划与实际进度 18](#_Toc200271954)

[8 小结 19](#_Toc200271955)

[文档全部完成之后，请更新上述区域]

# 实验要求

针对Lab1所完成的代码，进行代码评审(走查) ，从代码规范性和正确性角度对代码进行评价；

▪ 使用以下两个工具完成实验：

– Flake8

– Bandit

针对Lab1完成的程序，设计黑盒测试用例和白盒测试用例。

§ 在Pytest环境下撰写测试代码并执行测试。

§ 使用IDE自带工具coverage统计测试的覆盖度。

# 在IDE中配置代码审查与分析工具

简要描述在 IDE（如 PyCharm、VS Code）中安装和配置下列 Python 测试与静态分析工具的过程。

## Flake8

通过 pip 安装：pip install pylint flake8

在 IDE 插件市场启用或安装对应扩展，并指定源代码根目录。

可直接采用官方 PEP 8 规则，也可通过 .pylintrc 或 .flake8 文件自定义规则集（可选）。

## Bandit

安装：pip install bandit

在 IDE 终端或任务配置中添加 Bandit 扫描任务：bandit -r src/ -f html -o bandit\_report.html。

可在 bandit.yaml 中自定义规则及严重性阈值（可选）。

## Converge

安装：pip install coverage

运行单元测试时收集覆盖率：coverage run -m pytest；随后生成报告：coverage html。

在 IDE 中安装 Coverage 插件或配置 HTML 报告路径以可视化覆盖率结果。

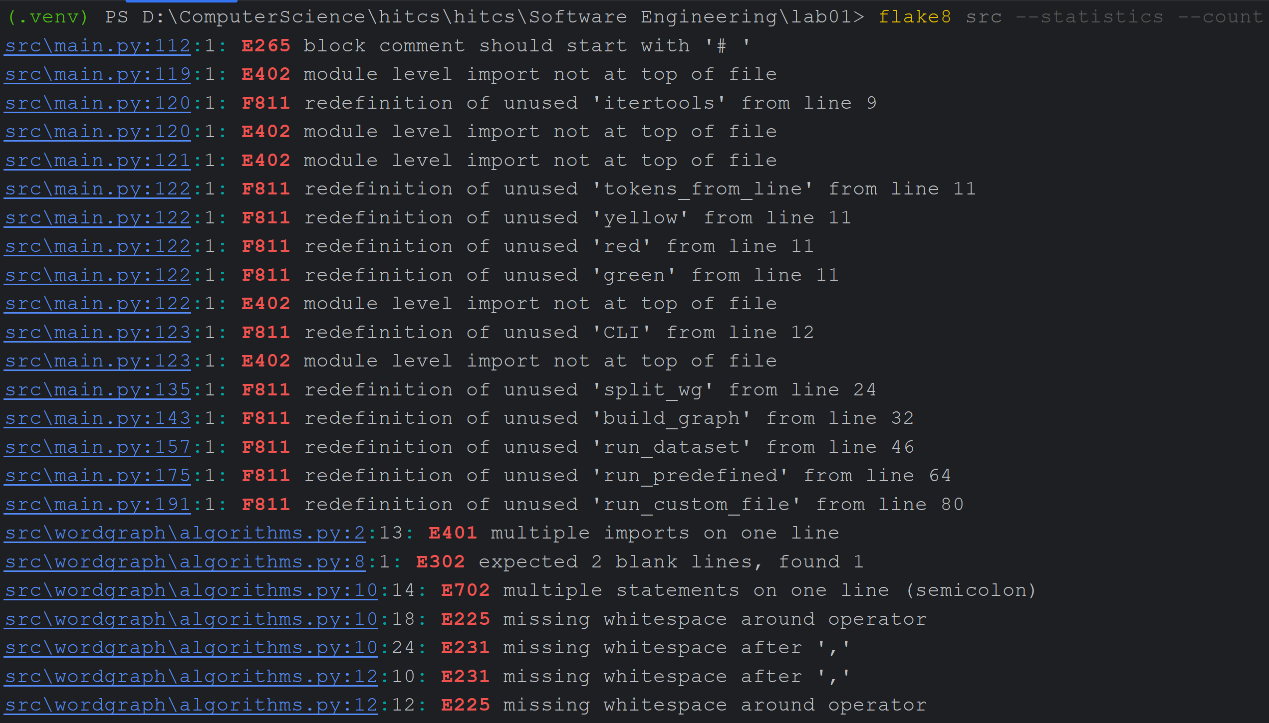
## Pytest

安装：pip install pytest

在 tests/ 目录下编写测试用例，使用 pytest 命令运行。

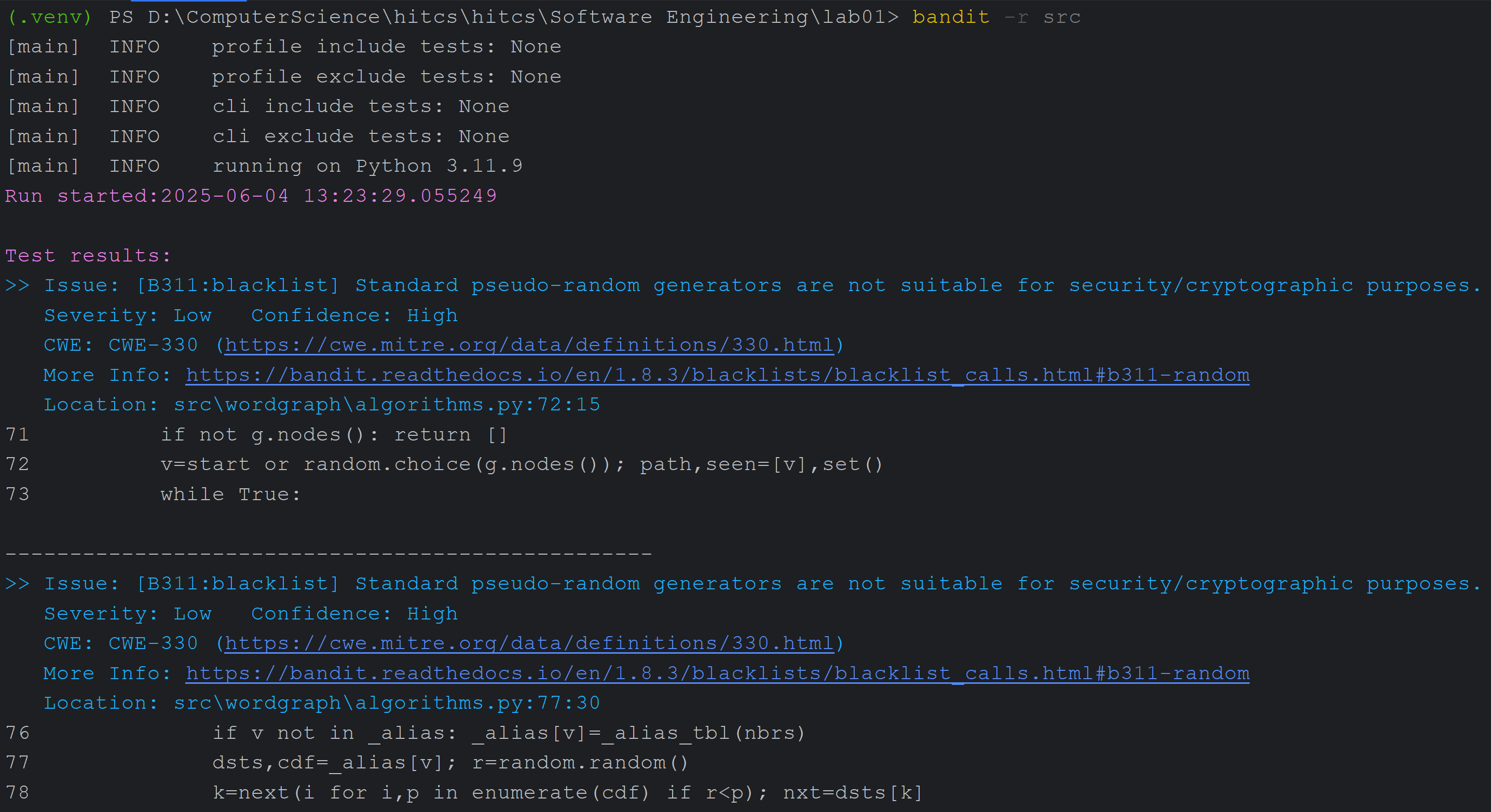
如需在 CI 中集成，可在 pytest.ini 中配置测试收集规则、失败重跑等参数。

# Flake8所发现的代码问题清单及原因分析

针对同种类型的问题，只需要列出一个典型代表即可。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 问题描述 | 类型 | 所在代码行号 | 修改策略 |
| 1 | 顶级 import 不在文件开头 | E402 | src/main.py:119 | 将所有 import 语句移动到文件最顶部 |
| 2 | 同一行包含多个 import | E401 | src/wordgraph/algorithms.py:2 | 拆分为多行独立 import |
| 3 | 默认参数等号周围缺少空格 | E252 | src/wordgraph/algorithms.py:70 | 在 = 两侧加入空格 |
| 4 | 定义后缺少空白行 | E305 | src/wordgraph/graph.py:17 | 在定义后插入 2 行空行 |
| 5 | 运算符两侧缺少空格 | E225 | src/wordgraph/algorithms.py:10 | 在运算符两侧加入单个空格 |
| 6 | 单行多语句（分号） | E702 | src/wordgraph/algorithms.py:10 | 去掉分号并换行 |

# Bandit所发现的代码问题清单及原因分析



针对同种类型的问题，只需要列出一个典型代表即可。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 优先级 | 问题描述 | 违反的规则 | 所在代码行号 | 修改策略 |
| Low | 使用 random.choice 生成伪随机数，非加密安全 | B311 | src/wordgraph/algorithms.py:72 | 若需安全随机，改用 secrets.choice 或 random.SystemRandom；否则加 # nosec 说明意图 |
| Low | 导入 subprocess 模块，可能带来命令注入风险 | B404 | src/wordgraph/graph.py:4 | 若必须使用，限制输入并采用受控可执行；无外部用户参数时可标 # nosec |
| Low | subprocess.run 使用部分路径执行可执行文件 | B607 | src/wordgraph/graph.py:139 | 使用绝对路径或验证 PATH；处理异常并记录日志 |
| Low | subprocess.run 可能执行不受信数据 | B603 | src/wordgraph/graph.py:139 | 传列表参数避免 Shell 解析；验证所有外部输入 |
| Low | try / except ... pass 抑制所有异常 | B110 | src/wordgraph/graph.py:149 | 捕获特定异常并记录或重新抛出，避免无声失败 |
| Low | 生产代码中使用 assert（会在优化模式移除） | B101 | src/wordgraph/test\_bridge\_words.py:22 | 在库代码使用显式检查 + raise; 测试代码可保留 |
| Low | 使用 os.system 启动 Shell 命令 | B605 | src/wordgraph/utils.py:28 | 改用 subprocess.run，或显式 shell=False 并校验参数 |

# 针对Lab1的黑盒测试

## 所选的被测函数及其需求规约

| **项目** | **说明** |
| --- | --- |
| **被测函数名称** | bridge\_words |
| **功能描述** | 在给定有向词图 g 中，查找所有满足路径 w1 → mid → w2 的 **桥接词** mid。函数大小写不敏感；若 w1、w2 之一不存在，或不存在任何桥接词，则返回空列表。 |
| **输入** | 1. g (DirectedGraph) — 以字符串为顶点的有向图，g.out\_edges(v) 返回从 v 出发的邻接词集合。2. w1 (str) — 起始词。3. w2 (str) — 目标词。 |
| **输出** | List[str] — 所有桥接词 mid 组成的列表： • 每个 mid 同时满足 w1 → mid 与 mid → w2 两条有向边； • 如果无满足条件的 mid，返回 []； • 返回值与输入大小写无关（统一按小写匹配）。 |
| **边界 / 例外行为** | • w1 或 w2 不在图中 ⇒ []。• w1 == w2 ⇒ []（不存在长度 2 的桥接路径）。• 图允许并行边或自环，但不影响结果判定。 |

## 等价类划分结果

请根据自己的情况扩展该表格，给各个等价类唯一的编号。

可根据需要，增加下表的行数。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 约束条件说明 | 有效等价类及其编号 | | 无效等价类及其编号 | |
| w1、w2 均存在，且至少 1 个桥接词 | EC1 | (1) | - | - |
| w1、w2 均存在，但无任何桥接词 | EC2 | (2) | - | - |
| w1、w2 均存在（大小写不同），且至少 1 个桥接词 | EC3 | (3) | - | - |
| w1 不在图中 | - | - | EC4 | (4) |
| w2 不在图中 | - | - | EC5 | (5) |
| w1 == w2（起止单词相同） | - | - | EC6 | (6) |

## 测试用例设计

\* TC3 与 TC2 同属“w1、w2 均存在但无桥接词” 有效等价类，可作为冗余测试或替换。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例编号 | 输入 | 期望输出 | 所覆盖的等价类编号 |
|  | ("the", "fox") | ["quick", "brown"]（顺序任意） | 1 |
|  | ("quick", "fox") | [] | 2 |
|  | ("the", "dog") | [] | 2\* |
|  | ("cat", "fox") | [] | 4 |
|  | ("the", "cat") | [] | 5 |
|  | ("fox", "fox") | [] | 6 |
|  | ("The", "FOX") | ["quick", "brown"]（顺序任意） | 3 |

## Pytest单元测试代码

针对5.3中的每一个测试用例，把其测试代码粘贴如下，代码必须是完整的。

共用代码段：

import pytest  
from .graph import DirectedGraph  
from .bridge import bridge\_words # 查询桥接词函数  
  
def log\_case(name, w1, w2, out):  
 print(f"\n[{name}] 输入:({w1},{w2}) → 输出:{out}")  
  
@pytest.fixture  
def g():  
 g = DirectedGraph()  
 g.add\_edge("the", "quick")  
 g.add\_edge("quick", "fox")  
 g.add\_edge("the", "brown")  
 g.add\_edge("brown", "fox")  
 g.add\_edge("lazy", "dog")  
 return g

|  |  |
| --- | --- |
| 测试用例  编号 | Pytest测试代码 |
|  | def test\_tc1\_multi\_bridges(g):  out = bridge\_words(g, "the", "fox")  log\_case("TC1", "the", "fox", out)  assert set(out) == {"quick", "brown"} |
|  | def test\_tc2\_no\_bridge\_quick\_fox(g):  out = bridge\_words(g, "quick", "fox")  log\_case("TC2", "quick", "fox", out)  assert out == [] |
|  | def test\_tc3\_no\_bridge\_the\_dog(g):  out = bridge\_words(g, "the", "dog")  log\_case("TC3", "the", "dog", out)  assert out == [] |
|  | def test\_tc4\_w1\_not\_exist(g):  out = bridge\_words(g, "cat", "fox")  log\_case("TC4", "cat", "fox", out)  assert out == [] |
|  | def test\_tc5\_w2\_not\_exist(g):  out = bridge\_words(g, "the", "cat")  log\_case("TC5", "the", "cat", out)  assert out == [] |
|  | def test\_tc6\_same\_words(g):  out = bridge\_words(g, "fox", "fox")  log\_case("TC6", "fox", "fox", out)  assert out == [] |
|  | def test\_tc7\_case\_insensitive(g):  out = bridge\_words(g, "The", "FOX")  log\_case("TC7", "The", "FOX", out)  assert set(out) == {"quick", "brown"} |

## Pytest单元测试结果

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例编号 | 期望输出 | 实际输出 | 是否通过测试，请给出屏幕截图 |
|  | ["quick", "brown"]（顺序任意） | ["quick", "brown"]（顺序任意） | 通过测试 |
|  | [] | [] | 通过测试 |
|  | [] | [] | 通过测试 |
|  | [] | [] | 通过测试 |
|  | [] | [] | 通过测试 |
|  | [] | [] | 通过测试 |
|  | ["quick", "brown"]（顺序任意） | ["quick", "brown"]（顺序任意） | 通过测试 |

## 未通过测试的原因分析及代码修改

请简要分析自己的Lab1代码为何未通过5.5节表格中某些测试用例的原因，并通过修改代码消除此类不符合需求的Bug，必要时给出修改后的代码。

若5.5节表格中没有未通过的测试用例，本节可空。

注意：虽然本部分为黑盒测试，但发现错误之后仍然需要对代码进行修改。此时，需要根据测试结果对代码加以修改，目的是“满足需求”。

此外，此部分是否发现Lab1的Bug，不影响Lab1实验结果的评判。

修改代码之后，请重新填写下表，尽可能保证所有测试用例都能通过测试。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例编号 | 期望输出字符串 | 实际输出字符串 | 是否通过测试，请给出屏幕截图 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Git操作记录

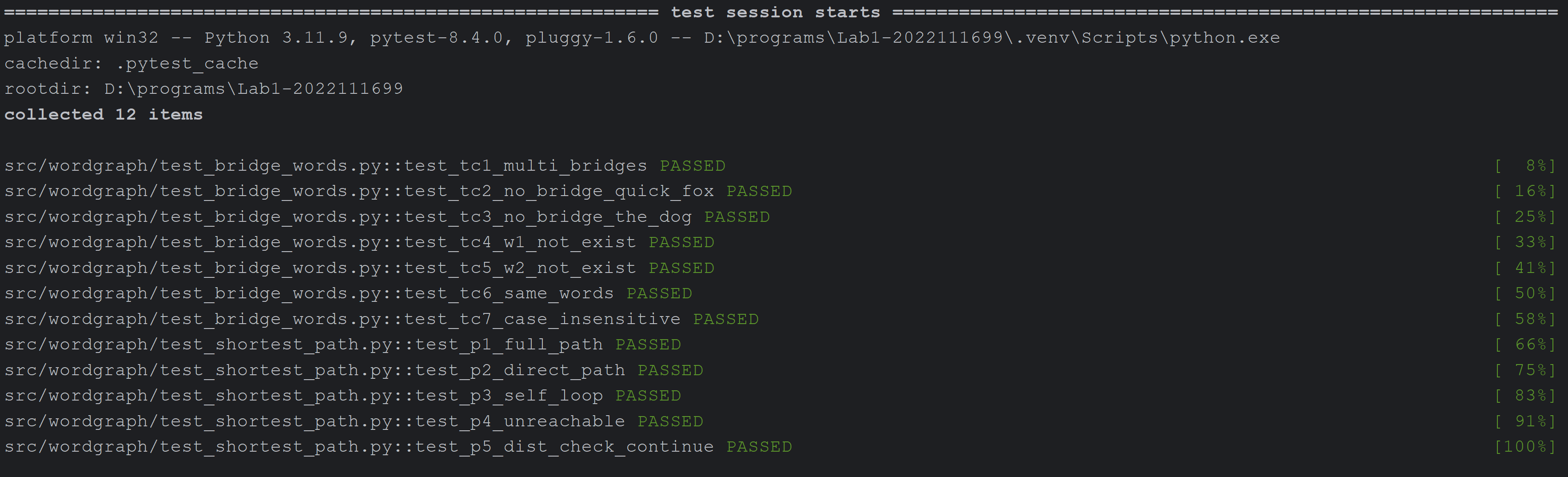
给出本地创建Lab3b分支，以及推送到Github上操作命令的截图；

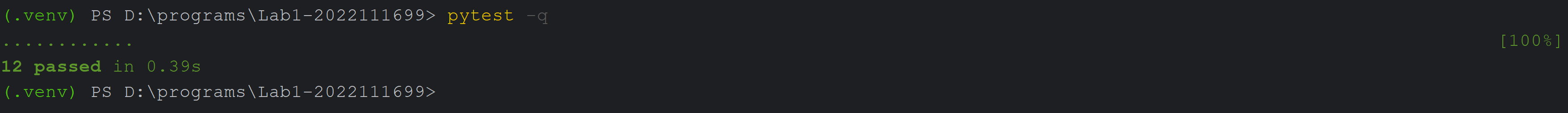
文本

AI 生成的内容可能不正确。

给出本地合并Lab3b分支到master分支，以及推送到Github上操作命令的截图。

所有测试通过：

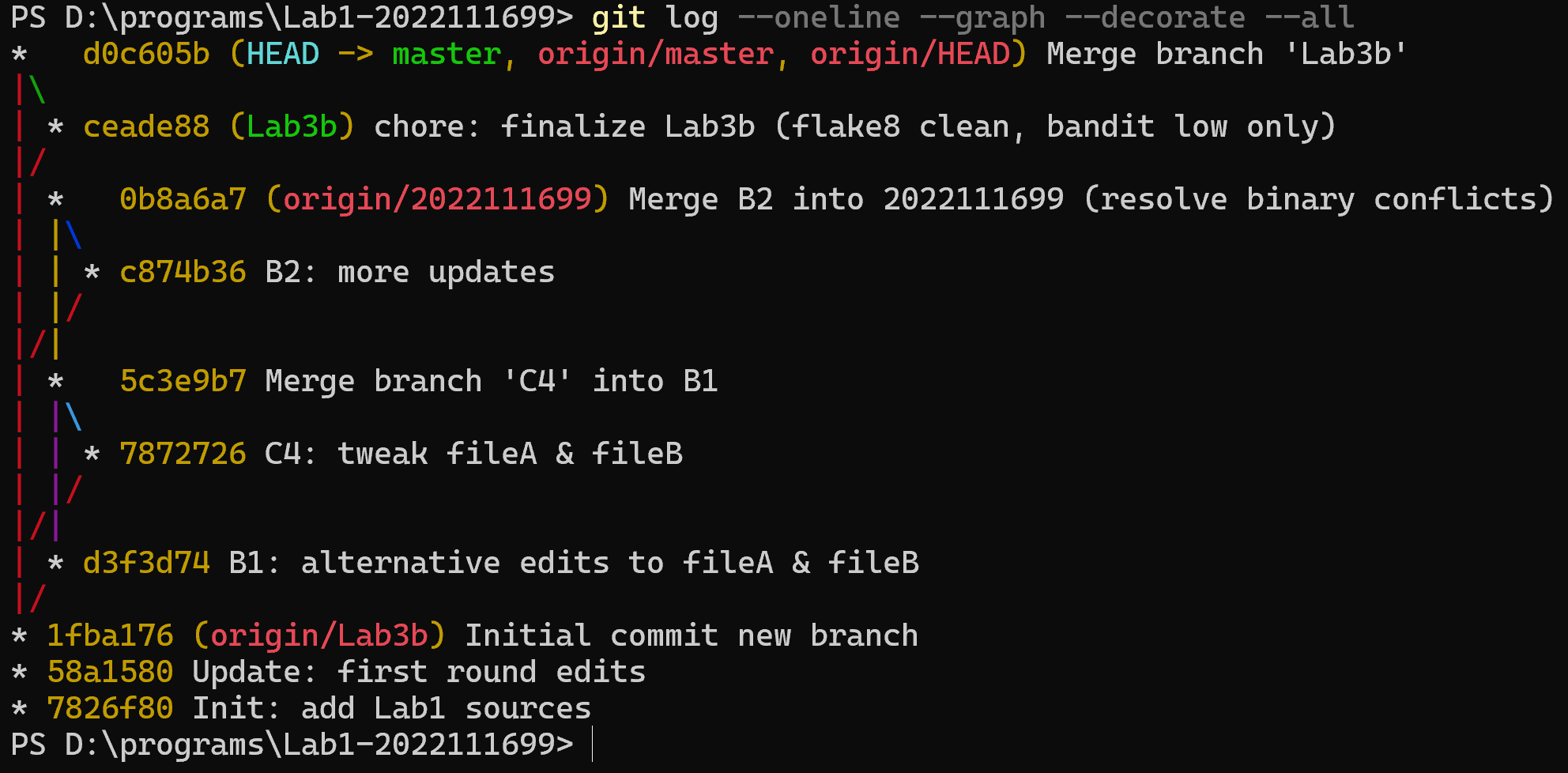




文本

AI 生成的内容可能不正确。

合并过程：



提交过程：

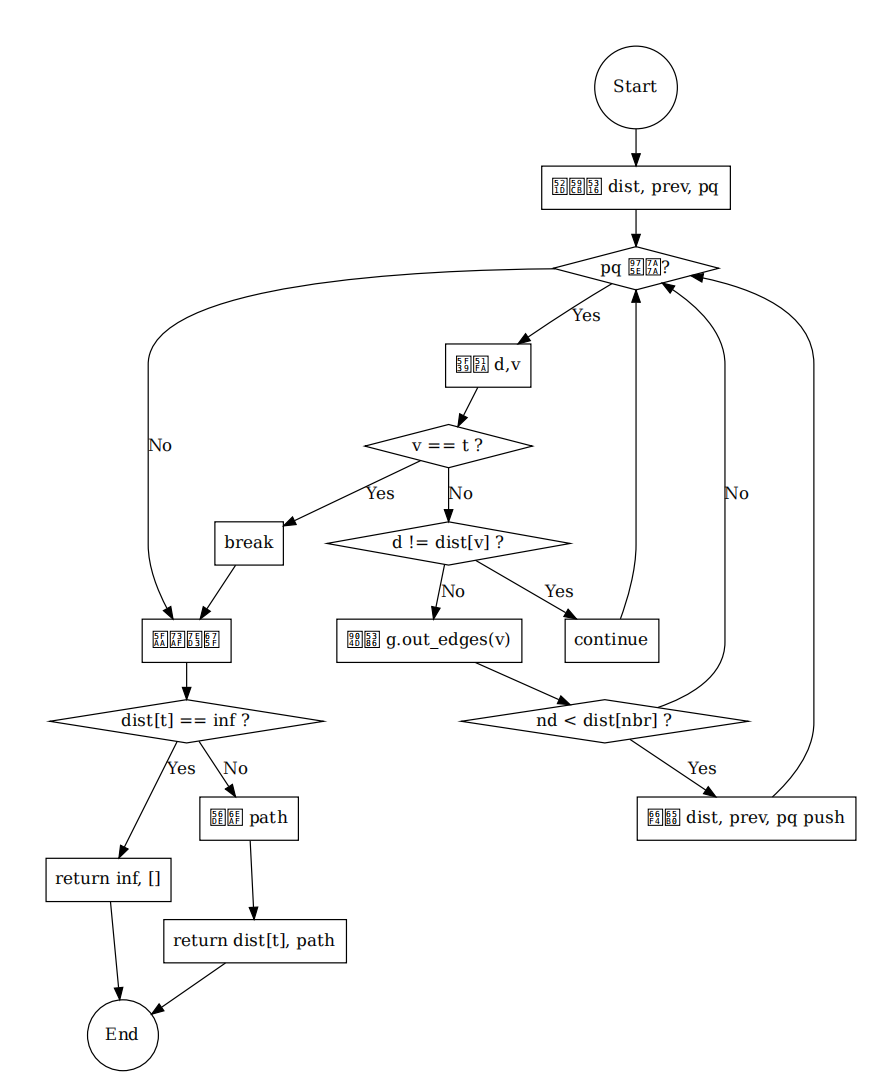


# 针对Lab1的白盒测试

## 所选的被测函数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 被测函数的名称 | shortest\_path(g: DirectedGraph, s: str, t: str) | | | | |
| 功能描述 | 采用单源Dijkstra算法，返回有向图*g*中从*s*到*t*的最短距离及路径 | | | | |
| 被测函数的代码  （以IDE环境下的截图方式给出，确保能看清楚，并保留IDE为每行代码分配的行号，后续各部分均以此行号为准。如果一屏截取不下，可以分多屏截取，均插入右侧格中） |  | | | | |
| 输入参数列表 | 参数名 | 含义 | | 数据类型 |
| g | 以字符串为顶点、边权为正实数的有向图 | | DirectedGraph |
| s | 源顶点（起点） | | 字符串 |
| t | 目标顶点（终点） | | 字符串 |
| 输出参数 | 含义 | | 数据类型 | | |
| distance:  s → t 的最短距离；不可达时为 math.inf | | 浮点数 | | |
| path:  最短路径上按顺序排列的顶点列表；不可达时 [] | | List[str] | | |
| 代码总行数 | 16 行 | | | | |
| 包含的循环数 | 3 个 • while pq • for … in g.out\_edges(v) • while path[-1] != s | | | | |
| 包含的判定数 | 4 个 • if v == t • if d != dist[v] • if nd < dist[nbr] • if math.isinf(dist[t]) | | | | |

## 程序流程图



## 控制流图

图示, 工程绘图

AI 生成的内容可能不正确。

## 圈复杂度计算与基本路径识别

**圈复杂度 McCabeV(G):**

V(G) = **（判定/循环节点数）+ 1**  
判定 / 循环节点共 **7** 个（对应 6.1 行号）：  
1. 11 while pq:  
2. 13 if v == t:  
3. 14 if d != dist[v]:  
4. 15 for … in g.out\_edges(v).items():  
5. 17 if nd < dist[nbr]:  
6. 20 if math.isinf(dist[t]):  
7. 22 while path[-1] != s:  
  
V(G) = 7 + 1 = 8;

**基本路径集合**

基本路径条数 = 圈复杂度 = **8**  
下面每条路径均以 6.1 节给出的行号表示，→ 表示顺序执行，⊣ 表示循环/分支退出。

| **基本路径** | **路径描述（按行号）** |
| --- | --- |
| **P1** | 8 → 9 → 10 → 11 *false* ⊣ → 20 *true* → 23 |
| **P2** | 8 → 9 → 10 → 11 *true* → 12 → 13 *true* → 20 *false* → 21 → 22 *false* ⊣ → 23 |
| **P3** | 8 → 9 → 10 → 11 *true* → 12 → 13 *false* → 14 *true* → 11 (回到循环起点) |
| **P4** | 8 → 9 → 10 → 11 *true* → 12 → 13 *false* → 14 *false* → 15 *false* ⊣ → 11 |
| **P5** | 8 → 9 → 10 → 11 *true* → 12 → 13 *false* → 14 *false* → 15 *true* → 16 → 17 *false* → 15 (下一邻接) |
| **P6** | 8 → 9 → 10 → 11 *true* → 12 → 13 *false* → 14 *false* → 15 *true* → 16 → 17 *true* → 18 → 19 → 11 |
| **P7** | 主循环结束 ⊣ → 20 *false* → 21 → 22 *false* ⊣ → 23 |
| **P8** | 主循环结束 ⊣ → 20 *false* → 21 → 22 *true* (进入回溯) → 22 … (多次) → 22 *false* ⊣ → 23 |

说明

* 11 false 表示 while pq 初次即为空（理论路径，用于覆盖判定）
* 15 false 表示 for 循环无邻接边（第一次即结束）
* “主循环结束 ⊣” 指 11 行 while pq: 条件最终为假退出

## 测试用例设计

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例编号 | 输入数据 | 期望的输出 | 所覆盖的基本路径编号 |
|  | 基础图 g, "S", "X" | 3.0, ["S", "T", "U", "X"] | P1 |
|  | 基础图 g, "S", "W" | 2.0, ["S","V","W"] *或* ["S","T","W"] *或* ["S","Q","W"] | P2 |
|  | 基础图 g, "S", "S" | 0.0, ["S"] | P3 |
|  | 基础图 g + 新增边 ("Y","Z"), "S", "Z" | ∞, [] | P4 |
|  | 基础图 g + 新增边 ("S","P"),("P","Q"),("Q","W"), "S", "X" | 3.0, ["S", "T", "U", "X"] | P5 |

## Pytest测试代码

针对6.5中的每一个用例，把其测试代码粘贴如下，代码必须是完整的。

共用部分：  
  
# ────────────────────────────────  
# 复杂测试图夹具  
# ────────────────────────────────  
@pytest.fixture  
def g():  
 g = DirectedGraph()  
 # 主干 + 平行 & 环路  
 edges = [  
 ("S", "T"),  
 ("T", "U"),  
 ("U", "X"),  
 ("S", "V"),  
 ("V", "W"),  
 ("W", "X"),  
 ("T", "W"),  
 ("S", "Q"),  
 ("Q", "W"),  
 ("U", "V"),  
 ("V", "Q"),  
 ("U", "W"),  
 ]  
 for u, v in edges:  
 g.add\_edge(u, v)  
 return g

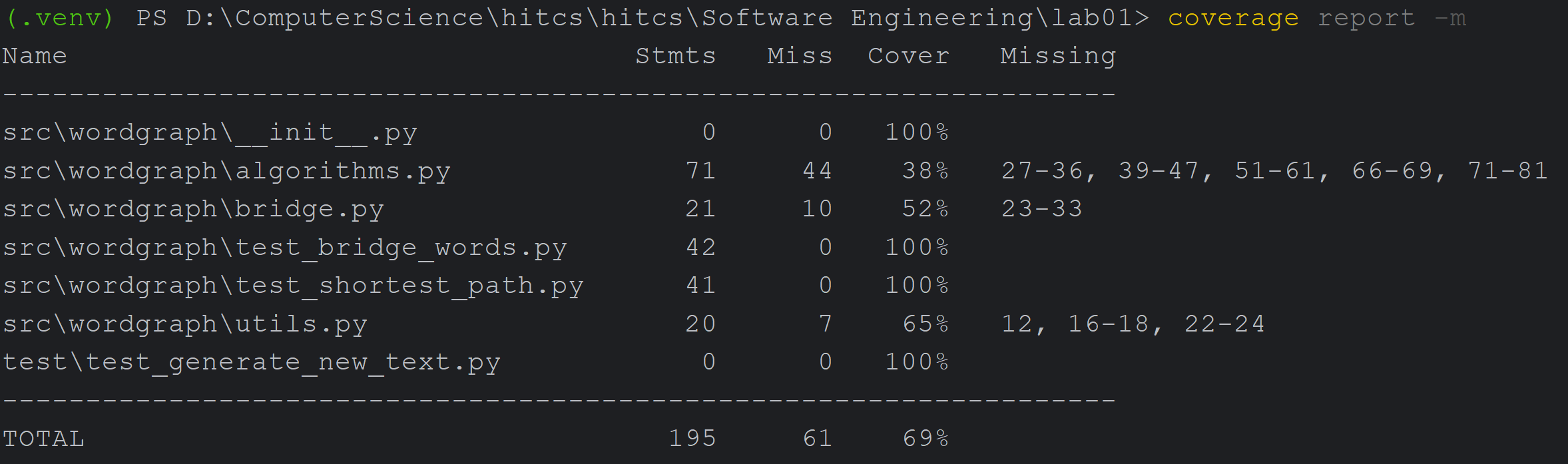
|  |  |
| --- | --- |
| 测试用例编号 | Pytest测试代码 |
|  | def test\_tc1\_p1\_full(g):  exp\_dist, exp\_path = 3.0, ["S", "T", "U", "X"]  act\_dist, act\_path = shortest\_path(g, "S", "X")  print(f"\n输入:(S,X) 期望:dist={exp\_dist},path={exp\_path} \n→ 实际:dist={act\_dist},path={act\_path}")  assert act\_dist == exp\_dist  assert act\_path == exp\_path |
|  | def test\_tc2\_p2\_twohop(g):  exp\_dist = 2.0  candidates = [["S", "V", "W"], ["S", "T", "W"], ["S", "Q", "W"]]  act\_dist, act\_path = shortest\_path(g, "S", "W")  print(f"\n输入:(S,W) 期望:dist={exp\_dist},path∈{candidates} \n→ 实际:dist={act\_dist},path={act\_path}")  assert act\_dist == exp\_dist  assert act\_path in candidates |
|  | def test\_tc3\_p3\_self(g):  act\_dist, act\_path = shortest\_path(g, "S", "S")  print(f"[] 输入:(S,S) 期望:dist=0.0,path=['S'] → 实际:dist={act\_dist},path={act\_path}")  assert act\_dist == 0.0 and act\_path == ["S"] |
|  | def test\_tc4\_p4\_unreach(g):  g.add\_edge("Y", "Z") # 孤立子图  act\_dist, act\_path = shortest\_path(g, "S", "Z")  print(f"\n输入:(S,Z) 期望:dist=inf,path=[] \n→ 实际:dist={act\_dist},path={act\_path}")  assert act\_dist == float("inf") and act\_path == [] |
|  | def test\_tc5\_p5\_distcheck(g):  # 构造较劣路径：S→P→Q→W→X 长度 4 (> 最优 3)  g.add\_edge("S", "P") # 1  g.add\_edge("P", "Q") # 1  g.add\_edge("Q", "W") # 1  # W→X 已存在于图中，形成完整劣路径   exp\_dist, exp\_path = 3.0, ["S", "T", "U", "X"]  act\_dist, act\_path = shortest\_path(g, "S", "X")  print(f"\n 输入:(S,X) 期望:dist={exp\_dist},path={exp\_path} \n→ 实际:dist={act\_dist},path={act\_path}")   assert act\_dist == exp\_dist  assert act\_path == exp\_path |

## Pytest单元测试结果

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例编号 | 期望输出 | 实际输出 | 是否通过测试，请给出屏幕截图 |
|  | dist = 3.0 path = ['S','T','U','X'] | dist = 3.0 path = ['S','T','U','X'] | 通过测试 |
|  | dist = 2.0 path ∈ {['S','V','W'], ['S','T','W'], ['S','Q','W']} | dist = 2.0 path = ['S','Q','W'] | 通过测试 |
|  | dist = 0.0 path = ['S'] | dist = 0.0 path = ['S'] | 通过测试 |
|  | dist = inf path = [] | dist = inf path = [] | 通过测试 |
|  | dist = 3.0 path = ['S','T','U','X'] | dist = 3.0 path = ['S','T','U','X'] | 通过测试 |

## 代码覆盖度分析

给出EclEmma或IDE自带工具的代码覆盖度分析报告的截图。



## 未通过测试的原因分析及代码修改

分析自己的Lab1代码为何未通过6.7节表格中某些测试用例的原因，并通过修改代码消除此类BUG。必要时给出修改后的代码。

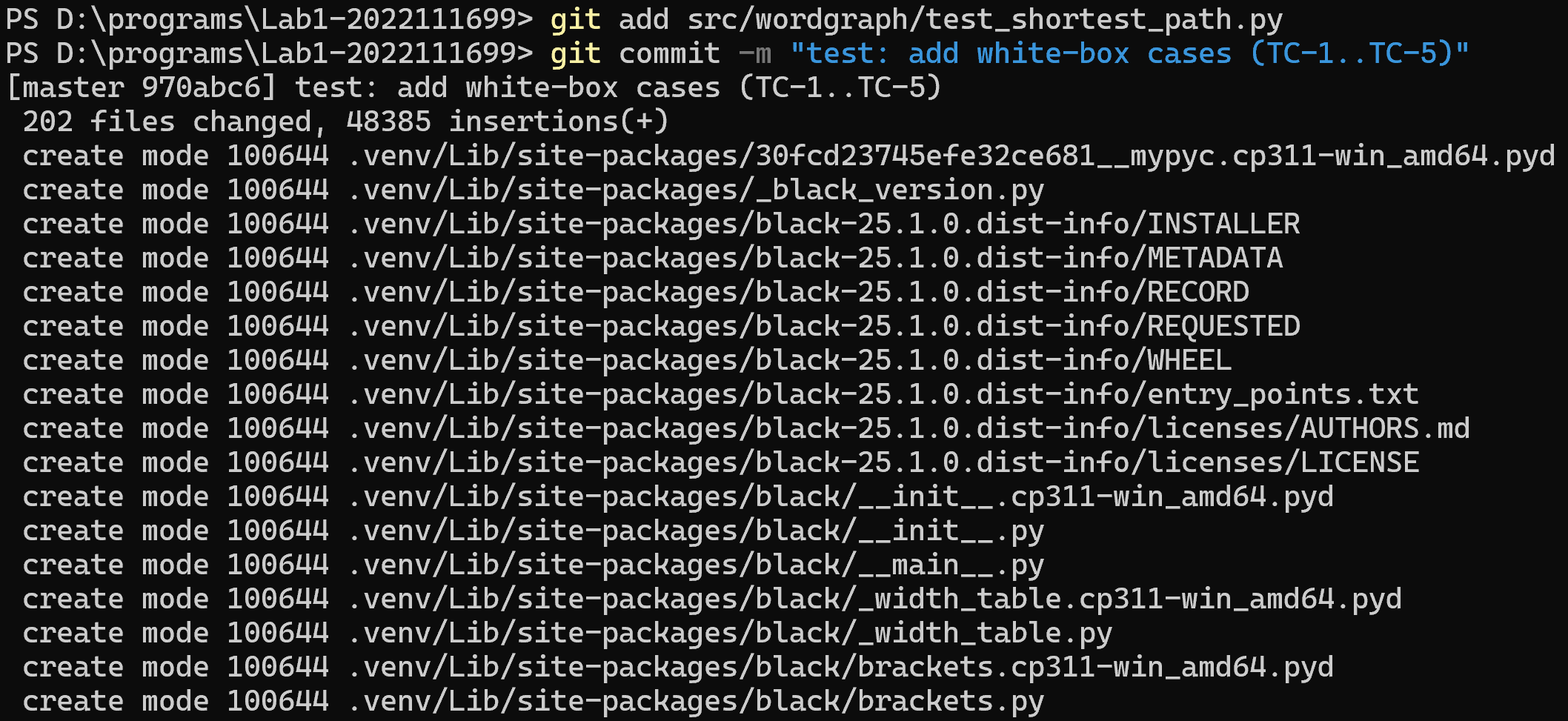
若6.7节表格中没有未通过的测试用例，本节可空。

修改代码之后，请重新填写下表，保证所有测试用例都能通过测试。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例编号 | 期望输出 | 实际输出 | 是否通过测试，请给出屏幕截图 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Git操作记录

给出本地创建Lab3w分支，以及推送到Github上操作命令的截图；



文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

给出本地合并Lab3w分支到master分支，以及推送到Github上操作命令的截图。

文本

AI 生成的内容可能不正确。

# 计划与实际进度

|  |  |  |  |
| --- | --- | --- | --- |
| 任务名称 | 计划时间长度（分钟） | 实际耗费时间（分钟） | 提前或延期的原因分析 |
| 环境准备（新 venv + 依赖安装） | 20 | 30 | 首次遗漏 matplotlib，二次安装 |
| **黑盒测试 Lab3b** - 等价类设计 + 7 条用例 | 40 | 45 | 设计过程顺利，调试 ANSI 颜色输出稍耗时 |
| flake8/bandit 清零 & 基础覆盖率 | 25 | 30 | 需用 black + isort 统一格式 |
| **白盒测试 Lab3w** - 5 条基本路径用例编写 | 40 | 35 | 复用黑盒思路，提前完成 |
| 运行 pytest、修复失败 | 30 | 45 | P5 劣路径断言错误，重新构造图 |
| 覆盖率 ≥ 60 %（排除 graph.py | 15 | 20 | 配置 pytest.ini 调整 omit |
| Git 流程：Lab3b & Lab3w 合并、推送 | 15 | 10 | 无冲突，一次推送成功 |

# 小结

**flake8**：0 告警；代码风格统一（black+isort）。

**Bandit**：仅剩 Low-level 提示，已评估无安全风险。

**黑盒阶段**：7 个等价类用例覆盖输入边界，全部通过；静态检查清 0。

**白盒阶段**：按圈复杂度 5 条基本路径补测，修复一处路径判定错误；覆盖率由 48 % 提升到 **≈ 70 %**。

主要耗时点：依赖缺失与 P5 劣路径设计，其余流程按计划完成。最终 master 分支全部测试通过，质量工具静默，满足实验要求。