AI1804 算法设计与分析
第 4 次课上练习：图搜索与拓扑排序

练习说明：

- 本次练习共 2 道题，总时间 90 分钟（两节课）
- 重点：图搜索算法的应用，拓扑排序的应用
- 每道题都需要用 Python 编程实现
- 注意理解图搜索算法的核心思想和应用场景

问题 4-1：社交网络中的最短路径与连通分量

问题描述

给定一个社交网络（无向图），每个顶点代表一个用户，边代表用户之间的好友关系。请实现以下功能：

任务 **1**：最短路径查找

- 给定两个用户 s 和 t，找出从 s 到 t 的最短路径
- 返回最短路径（顶点列表）和路径长度
- 如果不存在路径，返回 `(None, -1)`

任务 **2**：连通分量分析

- 找出所有连通分量（每个连通分量是一个用户群组）
- 返回每个连通分量中的用户列表

示例：

```
图结构（邻接表表示）：
graph = {
    'Alice': ['Bob', 'Charlie'],
    'Bob': ['Alice', 'David'],
    'Charlie': ['Alice', 'Eve'],
    'David': ['Bob'],
    'Eve': ['Charlie'],
    'Frank': ['Grace'],
    'Grace': ['Frank'],
    'Henry': []  # 孤立节点
}

任务1：shortest_path(graph, 'Alice', 'Eve')
输出：(['Alice', 'Charlie', 'Eve'], 2)
```

任务2: connected_components(graph)
输出: [['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
       ['Frank', 'Grace'],
       ['Henry']]

要求:

- 时间复杂度: $O(|V| + |E|)$
- 空间复杂度: $O(|V| + |E|)$

代码框架

```
def shortest_path(graph, start, target):
    """
    查找从start到target的最短路径

    参数:
        graph: 无向图, 邻接表表示 {vertex: [neighbors]}
        start: 起始顶点
        target: 目标顶点

    返回:
        (path, length) 元组, 其中:
        - path: 最短路径的顶点列表, 如果不存在路径则为None
        - length: 最短路径长度, 如果不存在路径则为-1

    时间复杂度: O(|V| + |E|)
    """
    # TODO: 实现算法

    pass

def connected_components(graph):
    """
    找出所有连通分量

    参数:
        graph: 无向图, 邻接表表示 {vertex: [neighbors]}

    返回:
        连通分量列表, 每个分量是一个顶点列表

    时间复杂度: O(|V| + |E|)
    """
    # TODO: 实现算法

    pass
```

```
# ========== 测试代码 ==========

print("=== 测试1: 最短路径查找 ===")
graph1 = {
    'Alice': ['Bob', 'Charlie'],
    'Bob': ['Alice', 'David'],
    'Charlie': ['Alice', 'Eve'],
    'David': ['Bob'],
    'Eve': ['Charlie'],
    'Frank': ['Grace'],
    'Grace': ['Frank'],
    'Henry': []
}

# 测试路径存在
path1, length1 = shortest_path(graph1, 'Alice', 'Eve')
print(f"Alice到Eve的最短路径: {path1}")
print(f"路径长度: {length1}")
assert path1 == ['Alice', 'Charlie', 'Eve'], f"路径不正确: {path1}"
assert length1 == 2, f"期望长度2, 实际{length1}"

# 测试路径不存在
path2, length2 = shortest_path(graph1, 'Alice', 'Frank')
print(f"Alice到Frank的最短路径: {path2}")
print(f"路径长度: {length2}")
assert path2 is None, f"期望None, 实际{path2}"
assert length2 == -1, f"期望-1, 实际{length2}"

# 测试相同顶点
path3, length3 = shortest_path(graph1, 'Alice', 'Alice')
print(f"Alice到Alice的最短路径: {path3}")
print(f"路径长度: {length3}")
assert path3 == ['Alice'], f"期望['Alice'], 实际{path3}"
assert length3 == 0, f"期望0, 实际{length3}"

print("  最短路径测试通过")

print("\n=== 测试2: 连通分量分析 ===")
components = connected_components(graph1)
print(f"连通分量数量: {len(components)}")
print("连通分量:")
for i, comp in enumerate(components, 1):
    print(f"  分量{i}: {sorted(comp)}")

# 验证结果
assert len(components) == 3, f"期望3个连通分量, 实际{len(components)}"
all_vertices = set()
for comp in components:
```

```
    all_vertices.update(comp)
assert all_vertices == set(graph1.keys()), "顶点集合不匹配"

print(" 连通分量测试通过")

print("\n=== 测试3: 边界情况 ===")
# 空图
empty_graph = {}
path_empty, length_empty = shortest_path(empty_graph, 'A', 'B')
assert path_empty is None and length_empty == -1
assert connected_components(empty_graph) == []

# 单顶点图
single_graph = {'A': []}
path_single, length_single = shortest_path(single_graph, 'A', 'A')
assert path_single == ['A'] and length_single == 0
components_single = connected_components(single_graph)
assert len(components_single) == 1 and components_single[0] == ['A']

print(" 边界测试通过")

print("\n" + "="*50)
print("所有测试通过! ")
print("="*50)
```

问题 **4-2**：课程依赖关系与拓扑排序

问题描述

某大学需要安排课程的学习顺序。每门课程可能有先修课程（prerequisites），即在学习某门课程之前必须先完成某些课程。

任务 **1**：拓扑排序

- 给定课程依赖关系（有向图），找出一个合法的学习顺序
- 如果存在循环依赖（环），返回 `None`

任务 **2**：课程学习计划

- 给定一个目标课程列表，找出学习这些课程所需的所有先修课程
- 返回一个包含所有必需课程（包括目标课程和先修课程）的拓扑排序

示例：

```
课程依赖关系（有向图）：
prerequisites = {
    '数据结构': ['程序设计基础'],
    '算法设计': ['数据结构', '离散数学'],
    '操作系统': ['数据结构', '计算机组成原理'],
    '编译原理': ['数据结构', '算法设计'],
    '数据库': ['数据结构'],
    '程序设计基础': [],
    '离散数学': [],
    '计算机组成原理': []
}

任务1：topological_sort(prerequisites)
输出：['程序设计基础', '离散数学', '计算机组成原理', '数据结构',
       '算法设计', '操作系统', '数据库', '编译原理']
（注意：可能有多种合法顺序）

任务2：course_plan(prerequisites, ['编译原理', '操作系统'])
输出：包含所有必需课程的拓扑排序
（需要包含：编译原理、操作系统及其所有先修课程）
```

要求：

- 如果存在环，必须能够检测并返回 `None`
- 时间复杂度：$O(|V| + |E|)$
- 空间复杂度：$O(|V| + |E|)$

代码框架

```
def topological_sort(graph):
    """
    对有向图进行拓扑排序

    参数:
        graph: 有向图，邻接表表示 {vertex: [dependencies]}
               例如 {'A': ['B', 'C']} 表示A依赖于B和C（B和C是A的先修）

    返回:
        拓扑排序列表，如果存在环则返回None

    时间复杂度: O(|V| + |E|)
    """
    # TODO: 实现拓扑排序

    pass

def course_plan(prerequisites, target_courses):
    """
    找出学习目标课程所需的所有课程及其拓扑排序

    参数:
        prerequisites: 课程依赖关系 {course: [prerequisite_courses]}
        target_courses: 目标课程列表

    返回:
        包含所有必需课程的拓扑排序列表，如果存在环则返回None

    时间复杂度: O(|V| + |E|)
    """
    # TODO: 实现课程学习计划

    pass

# ========== 测试代码 ==========

print("=== 测试1: 拓扑排序（无环） ===")
prerequisites1 = {
    '数据结构': ['程序设计基础'],
    '算法设计': ['数据结构', '离散数学'],
    '操作系统': ['数据结构', '计算机组成原理'],
    '编译原理': ['数据结构', '算法设计'],
    '数据库': ['数据结构'],
    '程序设计基础': [],
    '离散数学': [],
    '计算机组成原理': []
}
```

```
result1 = topological_sort(prerequisites1)
print(f"拓扑排序结果: {result1}")

# 验证: 检查每条边的顺序是否正确
if result1:
    pos = {course: i for i, course in enumerate(result1)}
    for course, deps in prerequisites1.items():
        for dep in deps:
            assert pos[dep] < pos[course], \
                f"错误: {dep}应该在{course}之前"
    print("  拓扑排序验证通过")
else:
    print("  拓扑排序失败（检测到环）")

print("\n=== 测试2: 检测环 ===")
prerequisites2 = {
    'A': ['B'],
    'B': ['C'],
    'C': ['A']  # 形成环: A -> B -> C -> A
}

result2 = topological_sort(prerequisites2)
print(f"拓扑排序结果: {result2}")
assert result2 is None, "应该检测到环"
print("  环检测测试通过")

print("\n=== 测试3: 课程学习计划 ===")
target = ['编译原理', '操作系统']
plan = course_plan(prerequisites1, target)
print(f"学习计划: {plan}")

# 验证: 目标课程应该在计划中
if plan:
    assert '编译原理' in plan
    assert '操作系统' in plan
    # 验证先修课程也在计划中
    assert '数据结构' in plan
    assert '程序设计基础' in plan
    assert '算法设计' in plan
    assert '计算机组成原理' in plan

    # 验证拓扑顺序
    pos = {course: i for i, course in enumerate(plan)}
    for course in plan:
        if course in prerequisites1:
            for dep in prerequisites1[course]:
                if dep in plan:
```

```
                    assert pos[dep] < pos[course], \
                        f"错误: {dep}应该在{course}之前"

    print("  课程学习计划验证通过")
else:
    print("  课程学习计划失败（检测到环）")

print("\n=== 测试4: 边界情况 ===")
# 空图
empty_graph = {}
assert topological_sort(empty_graph) == []

# 单顶点图
single_graph = {'A': []}
result_single = topological_sort(single_graph)
assert result_single == ['A']

# 无依赖的多个课程
independent = {
    'A': [],
    'B': [],
    'C': []
}
result_indep = topological_sort(independent)
assert len(result_indep) == 3
assert set(result_indep) == {'A', 'B', 'C'}

print("  边界测试通过")

print("\n" + "="*50)
print("所有测试通过! ")
print("="*50)
```

提交说明

- 将代码保存在单个 Python 文件中
- 确保所有测试用例通过
- 代码应有适当注释，说明算法思路
- 分析每个函数的时间复杂度和空间复杂度
- 思考：BFS 和 DFS 分别适用于什么场景?

思考题（选做）

1. 在问题 4-1 中，为什么使用 BFS 而不是 DFS 来查找最短路径?

2. 在问题 4-2 中，如果使用 BFS 而不是 DFS，能否实现拓扑排序？为什么？

3. Full-BFS 和 Full-DFS 在时间复杂度上有什么区别？为什么？

4. 在问题 4-1 中，路径重构的时间复杂度是多少？如何优化？