**广东海洋大学学生实验报告书（学生用表）**

**GDOU-B-11-112**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 图算法 | | | 课程名称 | | 算法设计与分析 | | | | 课程号 | | 32260035 |
| 学院(系) | 数学与计算机 | | 专业 | 物联网工程 | | | | 班级 | | 1224 | | |
| 学生姓名 | 黎川滔 | 学号 | 202211672411 | | 实验地点 | | 明德楼A1501 | | 实验日期 | | 2025.4.22 | |

# 实验目的

本次实验旨在通过实践掌握图算法的核心应用。我首先实现了基于DFS的有向图环路检测算法；随后通过Kahn算法完成拓扑排序，解决课程学习顺序规划问题；在强连通分量分析中，我运用Tarjan算法识别图中的紧密连接子图。最后通过查找关键连接的实践使我深入理解了图算法的实现细节，提升了将理论知识转化为解决实际问题能力，为后续复杂算法学习打下坚实基础。

# 实验环境

处理器：AMD Ryzen 5 6600H with Radeon Graphics。

LeetCode。

# 实验内容

## 3.1 实验任务1 有向图中环路的存在判断

给定一个有向图，图的顶点数量为 n，边的数量为 m。请使用深度优先搜索（DFS）算法判断该有向图中是否存在环路。具体如下：

（1）假设学生的学号为 student\_id，计算学号字符串中所有数字的和 sum\_digits。

（2）如果 sum\_digits 是偶数，则在判断环路存在的 DFS 算法中，从编号为 sum\_digits % n 的顶点开始进行搜索；如果 sum\_digits 是奇数，则从编号为 (n - 1 - sum\_digits % n) 的顶点开始进行搜索。

（3）实现一个函数 hasCycle(graph)，其中 graph 是表示有向图的邻接表（或者其他合适的数据结构），函数返回一个布尔值，表示图中是否存在环路。

（4）请分析本问题的特点，知识体系，应用等内容，写出自己的体会。此问不允许用智能助手生成。

def hasCycle(graph):

n = len(graph)

student\_id = "202211672411"

sum\_digits = sum(int(ch) for ch in student\_id)

if sum\_digits % 2 == 0:

start = sum\_digits % n

else:

start = (n - 1 - sum\_digits % n)

visited = [0] \* n

def dfs(v):

if visited[v] == 1:

return True

if visited[v] == 2:

return False

visited[v] = 1

for neighbor in graph.get(v, []):

if dfs(neighbor):

return True

visited[v] = 2

return False

if dfs(start):

return True

for i in range(n):

if visited[i] == 0:

if dfs(i):

return True

return False

graph1 = {

0: [1],

1: [2],

2: [0]

}

graph2 = {

0: [1],

1: [2],

2: []

}

print(hasCycle(graph1))

print(hasCycle(graph2))

有向图中环路检测是图论的一个经典的问题，在课程安排和死锁检测等工程问题上都有具体的应用。对于这个问题，代码的做法是使用深度优先搜索进行图的遍历。具体而言，开辟状态数组，0表示未访问，1表示访问中，2表示已经访问从而来检测回边。在上述代码中，首先从特殊计算的 start 节点开始DFS，然后检查所有未被访问的节点，如果任何DFS发现环，立即返回True，全部检查完毕没有发现环则返回False。

## 3.2 实验任务2 拓扑排序

（1）对于一个给定的有向无环图（DAG），使用 Kahn 算法实现拓扑排序。

自行制定若干门课程的先修、后修课程，在不破坏原有图结构的前提下，实现拓扑排序算法，并输出正确的学习顺序。

（2）请分析本问题的特点，知识体系，应用等内容，写出自己的体会。此问不允许用智能助手生成。

from collections import defaultdict, deque

def kahn\_topological\_sort(courses):

graph = defaultdict(list)

in\_degree = defaultdict(int)

for pre, course in courses:

graph[pre].append(course)

in\_degree[course] += 1

if pre not in in\_degree:

in\_degree[pre] = 0

queue = deque([course for course in in\_degree if in\_degree[course] == 0])

topo\_order = []

while queue:

course = queue.popleft()

topo\_order.append(course)

for neighbor in graph[course]:

in\_degree[neighbor] -= 1

if in\_degree[neighbor] == 0:

queue.append(neighbor)

if len(topo\_order) == len(in\_degree):

return topo\_order

else:

return []

ai\_courses = [

("Python编程基础", "数据结构与算法"),

("数据结构与算法", "人工智能导论"),

("人工智能导论", "机器学习"),

("人工智能导论", "计算机视觉"),

("机器学习", "深度学习"),

("深度学习", "自然语言处理"),

("概率论与统计", "机器学习"),

("线性代数", "深度学习"),

("计算机视觉", "图像识别")

]

order = kahn\_topological\_sort(ai\_courses)

print("推荐学习顺序：")

for i, course in enumerate(order, 1):

print(f"{i}. {course}")

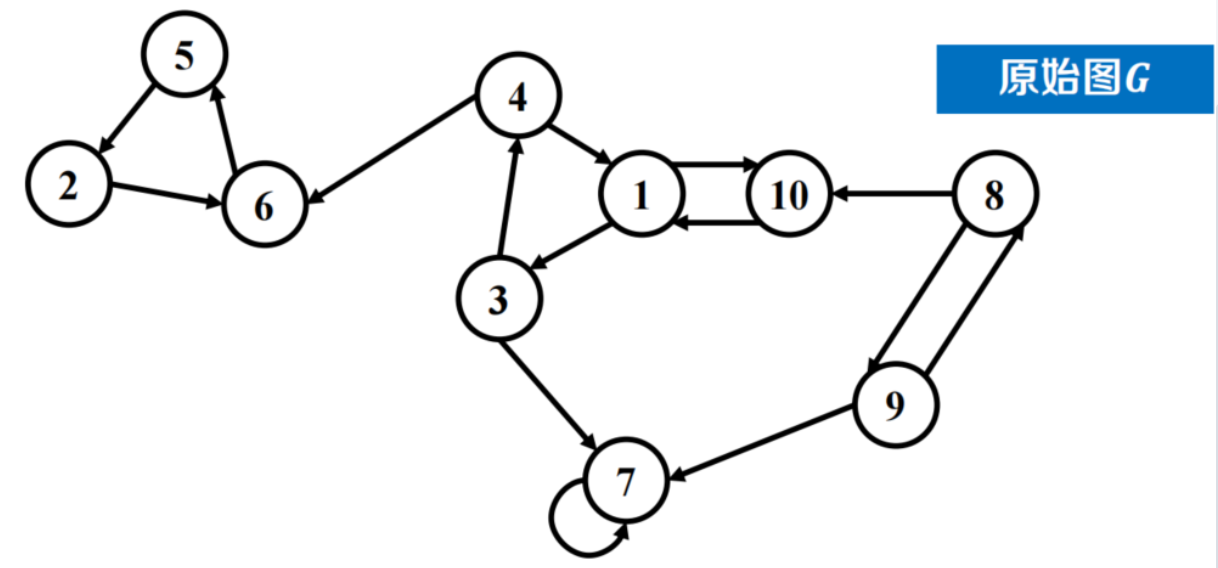
对于这个问题，在有向无环图中每一条边都表示一个先修关系。Kahn算法使用入度表和队列实现广度优先搜索，每次处理入度为0的节点。在上述代码中，使用 defaultdict(list) 表示的邻接表，存储图的边关系，使用 defaultdict(int) 记录每个顶点的入度，使用 deque 实现的队列，存储当前入度为0的顶点，使用 topo\_order 存储最终的拓扑排序结果。在拓扑排序主循环中，从队列中取出一个顶点，加入拓扑序列，遍历该顶点的所有后继顶点，将其入度减1，如果某个后继顶点的入度变为0，将其加入队列。输出结果为：

文本

AI 生成的内容可能不正确。

## 3.3 实验任务3 强连通分量

实现有向图G的强连通分量，请分析本问题的特点，知识体系，应用等内容，写出自己的体会。此问不允许用智能助手生成。



def tarjans\_scc(graph):

index = 0

stack = []

indices = {}

lowlink = {}

on\_stack = set()

result = []

def strongconnect(v):

nonlocal index

indices[v] = index

lowlink[v] = index

index += 1

stack.append(v)

on\_stack.add(v)

for w in graph.get(v, []):

if w not in indices:

strongconnect(w)

lowlink[v] = min(lowlink[v], lowlink[w])

elif w in on\_stack:

lowlink[v] = min(lowlink[v], indices[w])

if lowlink[v] == indices[v]:

# 找到一个强连通分量

scc = []

while True:

w = stack.pop()

on\_stack.remove(w)

scc.append(w)

if w == v:

break

result.append(scc)

for v in graph:

if v not in indices:

strongconnect(v)

return result

graph = {

1: [10],

2: [6],

3: [4, 7],

4: [1],

5: [6],

6: [3],

7: [9],

8: [9, 10],

9: [8],

10: [1]

}

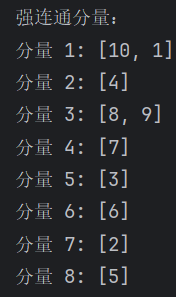
sccs = tarjans\_scc(graph)

print("强连通分量：")

for i, component in enumerate(sccs, 1):

print(f"分量 {i}: {component}")

在上述代码中，首先初始化节点，为节点分配索引和初始lowlink值以及将节点压入栈并标记为在栈中。如果邻居未被访问，递归处理，更新当前节点的lowlink值为自身和邻居lowlink的较小值；如果邻居在栈中，更新lowlink为自身和邻居索引的较小值。最终，当节点的lowlink等于其索引时，开始弹出栈中节点，直到弹回当前节点，这些节点构成一个强连通分量。



## 3.4 实验任务4 Leetcode 1192. 查找集群内的关键连接

from typing import List

class Solution:

def criticalConnections(self, n: int, connections: List[List[int]]) -> List[List[int]]:

graph = [[] for \_ in range(n)]

for u, v in connections:

graph[u].append(v)

graph[v].append(u)

discovery = [-1] \* n

low = [-1] \* n

time = 0

res = []

def dfs(u: int, parent: int):

nonlocal time

discovery[u] = low[u] = time

time += 1

for v in graph[u]:

if v == parent:

continue

if discovery[v] == -1:

dfs(v, u)

low[u] = min(low[u], low[v])

if low[v] > discovery[u]:

res.append([u, v])

else:

low[u] = min(low[u], discovery[v])

dfs(0, -1)

return res

在这段代码中，discovery 用于记录节点被访问的时间戳，low 用来记录节点通过非父节点边能回溯到的最早祖先的发现时间。首先将无向边转化为邻接表表示，接着进行初始化。在DFS函数中，为当前节点设置发现时间和初始low值，接着遍历所有邻居节点，对于未访问的子节点，递归调用DFS后更新low值，如果子节点的low值大于当前节点的发现时间，则该边为桥，对于已访问的节点，更新low值。

# 实验总结

我在这次实验的收获非常大，我学习并通过实践掌握了图相关的算法，尤其是有向图问题和强连通分量问题。除此之外，我通过几道老师布置的题目，掌握了深度优先搜索和迪杰斯特拉这些算法，在力扣平台的练习中进一步巩固。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 成绩 |  | 指导教师 |  | 日期 |  |