# Bài 15: Thực hành duyệt đồ thị theo chiều sâu

**Giải Chuyên đề Tin học 12 Bài 15: Thực hành duyệt đồ thị theo chiều sâu**  
**Khởi động trang 72 Chuyên đề Tin học 12**: Trong lí thuyết đồ thị, chu trình được định nghĩa là một đường đi không tầm thường khép kín, tức là đường đi có số cạnh lớn hơn 1 và đỉnh xuất phát trùng với đỉnh kết thúc. Làm cách nào để kiểm tra một đồ thị cho trước có chu trình hay không?  
**Lời giải:**  
Để kiểm tra xem một đồ thị có chu trình hay không, chúng ta có thể sử dụng thuật toán DFS (Duyệt Đầu Tiên) hoặc BFS (Duyệt Theo Chiều Rộng). Dưới đây là cách kiểm tra bằng DFS:  
- Bắt đầu từ một đỉnh bất kỳ trong đồ thị.  
- Thực hiện duyệt DFS từ đỉnh này.  
- Trong quá trình duyệt, nếu đỉnh nào đã được thăm trước đó và không phải là đỉnh cha của đỉnh hiện tại (trong trường hợp của cây), tức là tìm thấy một chu trình.  
- Nếu tất cả các đỉnh đều đã được duyệt và không tìm thấy chu trình, đồ thị không chứa chu trình.  
**Luyện tập 1 trang 74 Chuyên đề Tin học 12**: Hàm kiểm tra chu trình của đồ thị trên còn đúng không nếu đồ thị ban đầu là vô hướng?  
**Lời giải:**  
Để kiểm tra chu trình trong một đồ thị vô hướng, cách tiếp cận có thể khác so với đồ thị có hướng. Trong trường hợp đồ thị vô hướng, mỗi cạnh được coi là một liên kết giữa hai đỉnh, không có khái niệm đỉnh cha và đỉnh con.  
Một cách để kiểm tra chu trình trong đồ thị vô hướng là sử dụng duyệt DFS nhưng cần phải theo dõi đỉnh trước đó mà không phải là đỉnh hiện tại (cha của đỉnh hiện tại) để tránh lặp lại qua cùng một cạnh. Nếu trong quá trình duyệt, gặp một đỉnh đã được thăm trước đó và không phải là đỉnh cha của đỉnh hiện tại, thì có thể kết luận rằng đồ thị chứa chu trình.  
**Luyện tập 2 trang 74 Chuyên đề Tin học 12**: Viết lại hàm kiểm tra chu trình DFS\_acyclic(Adj,s) trong chương trình trên nhưng sử dụng phương án không đệ quy của thuật toán DFS.  
**Lời giải:**  
Phiên bản mẫu của hàm DFS\_acyclicsử dụng phương pháp không đệ quy của thuật toán DFS để kiểm tra xem một đồ thị có chứa chu trình hay không:  
def DFS\_acyclic(Adj, s):  
 stack = [(s, None)] # Dùng stack để thực hiện DFS, cặp (đỉnh, đỉnh cha)  
 visited = set() # Dùng set để theo dõi các đỉnh đã thăm  
 while stack:  
 node, parent = stack.pop() # Lấy đỉnh ra khỏi stack  
 if node in visited: # Nếu đỉnh đã được thăm trước đó  
 return True # Tìm thấy chu trình  
 visited.add(node) # Đánh dấu đỉnh đã thăm  
 for neighbor in Adj[node]: # Duyệt các đỉnh kề của đỉnh hiện tại  
 if neighbor != parent: # Tránh quay lại đỉnh cha  
 stack.append((neighbor, node)) # Thêm đỉnh kề vào stack  
 return False # Không tìm thấy chu trình  
# Sử dụng hàm kiểm tra chu trình không đệ quy  
graph\_undirected = {  
 0: [1, 2],  
 1: [0, 3],  
 2: [0, 3],  
 3: [1, 2]  
}  
if DFS\_acyclic(graph\_undirected, 0):  
 print("Đồ thị vô hướng có chu trình")  
else:  
 print("Đồ thị vô hướng không có chu trình")  
Trong hàm này:  
- Chúng ta sử dụng một stack để thực hiện duyệt DFS thay vì đệ quy.  
- Mỗi lần duyệt, chúng ta kiểm tra xem đỉnh hiện tại đã được thăm trước đó chưa. Nếu có, chúng ta tìm thấy chu trình.  
- Nếu không, chúng ta đánh dấu đỉnh đó đã được thăm và duyệt qua tất cả các đỉnh kề của nó, tránh quay lại đỉnh.  
**Vận dụng 1 trang 74 Chuyên đề Tin học 12**: Sửa lại chương trình bổ sung thông báo nếu hệ thống chuyên đề không hợp lí thì thông báo dãy các chuyên đề có mâu thuẫn về kiến thức.  
**Lời giải:**  
Phiên bản mẫu gợi ý sửa đổi của chương trình để bổ sung thông báo nếu hệ thống chuyên đề không hợp lý và hiển thị dãy các chuyên đề có mâu thuẫn về kiến thức:  
def DFS(graph, start, visited, stack):  
 visited[start] = True  
 for neighbor in graph[start]:  
 if not visited[neighbor]:  
 DFS(graph, neighbor, visited, stack)  
 stack.append(start)  
def topological\_sort(graph):  
 num\_nodes = len(graph)  
 visited = [False] \* num\_nodes  
 stack = []  
 for node in range(num\_nodes):  
 if not visited[node]:  
 DFS(graph, node, visited, stack)  
 return stack[::-1]  
def is\_valid\_system(topological\_order, prerequisites):  
 knowledge = set()  
 for course in topological\_order:  
 knowledge.add(course)  
 if course in prerequisites:  
 for prerequisite in prerequisites[course]:  
 if prerequisite not in knowledge:  
 return False, [course, prerequisite]  
 return True, None  
def check\_course\_system(prerequisites):  
 graph = {}  
 for course, prereq in prerequisites.items():  
 if course not in graph:  
 graph[course] = []  
 graph[course].extend(prereq)  
 topological\_order = topological\_sort(graph)  
 valid, conflicting\_topics = is\_valid\_system(topological\_order, prerequisites)  
 if valid:  
 print("Hệ thống chuyên đề hợp lý.")  
 else:  
 print("Hệ thống chuyên đề không hợp lý.")  
 print("Dãy các chuyên đề có mâu thuẫn về kiến thức:", conflicting\_topics)  
# Dữ liệu chuyên đề và tiên điều kiện  
prerequisites = {  
 "A": ["B"],  
 "B": ["C"],  
 "C": ["D"],  
 "D": ["A"] # Mâu thuẫn về kiến thức với tiên điều kiện của D  
}  
# Kiểm tra hệ thống chuyên đề  
check\_course\_system(prerequisites)  
**Vận dụng 2 trang 74 Chuyên đề Tin học 12**: Mở rộng bài tập trên cho đồ thị có hướng bất kì G = (V, E), được biểu diễn bởi ma trận kề A hoặc danh sách kề Adj. Viết hàm kiểm tra xem đồ thị G có chu trình hay không, nếu có thì hiển thị trên màn hình chu trình đó, bao gồm dãy các đỉnh tham gia vào chu trình.  
**Lời giải:**  
Hướng dẫn phiên bản mẫu của hàm để kiểm tra xem một đồ thị có chu trình hay không, và nếu có, hiển thị chu trình đó:  
def has\_cycle\_directed(graph):  
 def dfs(node, visited, stack):  
 visited[node] = True  
 stack.append(node)  
 for neighbor in graph[node]:  
 if neighbor in stack: # Nếu neighbor đã có trong stack, tức là tìm thấy chu trình  
 cycle\_start = stack.index(neighbor)  
 return stack[cycle\_start:]  
 if not visited[neighbor]: # Nếu neighbor chưa được thăm, tiếp tục duyệt DFS từ neighbor  
 result = dfs(neighbor, visited, stack)  
 if result: # Nếu tìm thấy chu trình từ neighbor, trả về chu trình  
 return result  
 stack.pop() # Loại bỏ node khỏi stack khi đã duyệt xong tất cả các neighbor của nó  
 return None  
 num\_nodes = len(graph)  
 visited = [False] \* num\_nodes  
 for node in range(num\_nodes):  
 cycle = dfs(node, visited, [])  
 if cycle: # Nếu tìm thấy chu trình từ đỉnh node, trả về chu trình  
 return cycle  
 return None  
def get\_cycle\_nodes(cycle, graph):  
 cycle\_nodes = []  
 for node in cycle:  
 cycle\_nodes.append(node)  
 if node == cycle[-1]: # Nếu đỉnh hiện tại là đỉnh cuối cùng của chu trình  
 break  
 return cycle\_nodes  
def check\_and\_print\_cycle(graph):  
 cycle = has\_cycle\_directed(graph)  
 if cycle:  
 cycle\_nodes = get\_cycle\_nodes(cycle, graph)  
 print("Đồ thị có chu trình:", cycle\_nodes)  
 else:  
 print("Đồ thị không có chu trình")  
# Hàm kiểm tra và hiển thị chu trình trong đồ thị  
graph\_directed = {  
 0: [1],  
 1: [2],  
 2: [3],  
 3: [4],  
 4: [2] # Chu trình: 2 -> 3 -> 4 -> 2  
}  
check\_and\_print\_cycle(graph\_directed)  
Trong mã này:  
- Hàm has\_cycle\_directed sử dụng duyệt DFS để kiểm tra xem đồ thị có chu trình hay không. Nếu tìm thấy chu trình, nó trả về chu trình dưới dạng một danh sách các đỉnh.  
- Hàm get\_cycle\_nodes giúp lấy ra danh sách các đỉnh tham gia vào chu trình từ chu trình được trả về bởi hàm has\_cycle\_directed.  
- Hàm check\_and\_print\_cycle kiểm tra và hiển thị chu trình trong đồ thị, nếu có