**Câu 1:**

**Kết quả:**

A screenshot of a computer

Description automatically generated

**Đồ thị:**

A diagram of a diagram

Description automatically generated

**Nhận xét:**

Nhận xét về khả năng liên thông từ đỉnh A trong đồ thị vô hướng:

1. Từ A đến B:
   * Có cạnh giữa A và B với trọng số 7. Do đồ thị vô hướng, nên từ A có thể đi đến B, và ngược lại từ B đến A.
2. Từ A đến C:
   * Có cạnh giữa A và C với trọng số 10. Do đồ thị vô hướng, nên từ A có thể đi đến C.
3. Từ A đến E:
   * Không có cạnh trực tiếp giữa A và E. Tuy nhiên, vì đồ thị vô hướng, từ A có thể đi đến B (cạnh AB), rồi từ B có thể đi đến E (cạnh BE). Do đó, có đường gián tiếp từ A đến E qua B.
4. Từ A đến D:
   * Từ A có thể đi đến B, từ B có cạnh nối tới D (cạnh BD). Do đó, có đường gián tiếp từ A đến D qua B.
5. Từ A đến Z:
   * Từ A có thể đi tới B, từ B đến E, và từ E có cạnh nối tới Z (cạnh EZ). Vì vậy, có đường gián tiếp từ A đến Z qua các đỉnh B và E.

Kết luận:

* Trong đồ thị vô hướng, từ đỉnh A, có thể đi tới B, C, E, D, và Z qua các con đường trực tiếp hoặc gián tiếp.
* Như vậy, đồ thị này là liên thông (có thể đi từ A đến mọi đỉnh khác), vì từ A có thể đi tới tất cả các đỉnh còn lại.

Vì đồ thị là vô hướng, tất cả các đỉnh đều có khả năng liên thông với nhau trực tiếp hoặc gián tiếp.

**Code:**

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self):

        self.graph = defaultdict(list)

    def addEdge(self, u, v):

        self.graph[u].append(v)

    def BFS(self, u):

        visited = [False] \* (len(self.graph) + 1)  # Adjusted for 1-based index

        stack = []

        visited[u] = True

        stack.append(u)

        while stack:

            u = stack.pop(0) #doi neu lam theo chieu sau

            print(u, end=' ')

            for i in self.graph[u]:

                if not visited[i]:

                    stack.append(i)

                    visited[i] = True

    def show(self):

        for i in self.graph:

            print(self.graph[i])

if \_\_name\_\_ == '\_\_main\_\_':

        g = Graph()

        g.addEdge(0, 1)

        g.addEdge(1, 0)

        g.addEdge(0, 2)

        g.addEdge(2, 0)

        g.addEdge(2, 4)

        g.addEdge(4, 2)

        g.addEdge(1, 4)

        g.addEdge(4, 1)

        g.addEdge(1, 3)

        g.addEdge(3, 1)

        g.addEdge(3, 4)

        g.addEdge(4, 3)

        g.addEdge(3, 5)

        g.addEdge(5, 3)

        g.addEdge(5, 4)

        g.addEdge(4, 5)

        print("BFS starting from vertex 0:")

        g.BFS(0)

**Câu 2:**

**Kết quả:**

A screenshot of a computer

Description automatically generated

**Nhận xét:**

**Khi cho 3 quân hậu với bàn cờ 3x3 thì kết quả:**

* **Không có lời giải hợp lệ**. Vì kích thước bàn cờ quá nhỏ và không có cách nào để đặt 3 quân hậu sao cho không có quân nào nằm cùng hàng, cột hoặc đường chéo.
* Chương trình sẽ backtrack qua các trạng thái khác nhau nhưng không tìm thấy lời giải, và tổng số lời giải sẽ là **0**.

**Code:**

import numpy as np

# Kiểm tra trạng thái hiện tại có phải là lời giải hợp lệ không

def is\_valid\_state(state, num\_queens):

return len(state) == num\_queens

# Tìm các vị trí hợp lệ tiếp theo cho quân hậu

def get\_candidates(state, num\_queens):

if not state:

return range(num\_queens)

position = len(state)

candidates = set(range(num\_queens))

for row, col in enumerate(state):

candidates.discard(col)

dist = position - row

candidates.discard(col + dist)

candidates.discard(col - dist)

return candidates

# Đệ quy thực hiện tìm kiếm theo chiều sâu để tìm các lời giải hợp lệ

def search(state, solutions, num\_queens):

if is\_valid\_state(state, num\_queens):

solutions.append(state.copy())

print(f"Valid State Found: {state}")

for candidate in get\_candidates(state, num\_queens):

state.append(candidate)

search(state, solutions, num\_queens)

print(f"Backtracking from: {state}")

state.remove(candidate)

# Chạy chương trình tìm lời giải

if \_\_name\_\_ == "\_\_main\_\_":

num\_queens = 8

solutions = []

search([], solutions, num\_queens)

print(f"Total Solutions: {len(solutions)}")

import numpy as np

def solve(num\_queens):

solutions = []

state = []

search(state, solutions, num\_queens)

return solutions

if \_\_name\_\_ == "\_\_main\_\_":

num\_queens = int(input("Enter number of queens: "))

solutions = solve(num\_queens)

for solution in solutions:

board = np.full((num\_queens, num\_queens), "-")

for row, col in enumerate(solution):

board[row][col] = 'Q'

print(f'\nSolution: {solution}')

print(board)