```
import tkinter as tk
from tkinter import ttk
import numpy as np
import networkx as nx
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg
import matplotlib.pyplot as plt

# Giá trị vô cực (đại diện cho đường không kết nối)
INF = float('inf')

# Hàm thực hiện giải thuật Floyd-Warshall
def floyd_warshall(graph):
    n = len(graph)
    dist = np.copy(graph)
    next_node = np.full((n, n), None)

    # Khởi tạo next_node cho các cạnh trực tiếp
    for i in range(n):
        for j in range(n):
            if graph[i][j] != INF and i != j:
                next_node[i][j] = j

    # Thực hiện giải thuật Floyd-Warshall
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
                    next_node[i][j] = next_node[i][k]

    return dist, next_node

# Hàm tái dựng đường đi ngắn nhất
def reconstruct_path(next_node, start, end):
    if next_node[start][end] is None:
        return []
    path = [start]
    while start != end:
        start = next_node[start][end]
        path.append(start)
    return path

# Hàm tạo đồ thị mẫu
def create_sample_graph(num_vertices):
```

```
    graph = np.full((num_vertices, num_vertices), INF)
    np.fill_diagonal(graph, 0)  # Khoảng cách từ đỉnh đến
chính nó là 0
    for i in range(num_vertices):
        for j in range(num_vertices):
            if i != j and np.random.rand() > 0.5:  # Xác
suất tạo cạnh
                graph[i][j] = np.random.randint(1, 10)  #
Trọng số ngẫu nhiên từ 1 đến 9
    return graph

# Hàm cập nhật đồ thị khi thay đổi số đỉnh
def update_graph_vertices():
    global graph, dist, next_node
    try:
        num_vertices = int(vertex_entry.get())
        if num_vertices < 2 or num_vertices > 10:  # Giới
hạn số lượng đỉnh
            raise ValueError("Số đỉnh phải từ 2 đến 10.")
    except ValueError:
        result_label.config(text="Vui lòng nhập số đỉnh
hợp lệ (2-10).")
        return

    # Tạo đồ thị mẫu mới
    graph = create_sample_graph(num_vertices)
    dist, next_node = floyd_warshall(graph)

    # Reset các ô nhập liệu và kết quả
    start_entry.delete(0, tk.END)
    start_entry.insert(0, "0")
    end_entry.delete(0, tk.END)
    end_entry.insert(0, str(num_vertices - 1))
    result_label.config(text="")

    # Hiển thị ma trận trọng số để chỉnh sửa
    show_weight_matrix(graph)

    # Vẽ đồ thị mới
    draw_graph(canvas_frame, graph)

# Hàm hiển thị ma trận trọng số để chỉnh sửa
def show_weight_matrix(graph):
    for widget in matrix_frame.winfo_children():
        widget.destroy()  # Xóa nội dung cũ
```

```
    rows, cols = graph.shape
    for i in range(rows):
        for j in range(cols):
            value = "" if graph[i][j] == INF else
int(graph[i][j])
            entry = ttk.Entry(matrix_frame, width=5,
justify="center")
            entry.grid(row=i, column=j, padx=5, pady=5)
            entry.insert(0, value)
            entry.bind("<FocusOut>", lambda e, i=i, j=j:
update_edge_weight(i, j, e.widget))

# Hàm cập nhật trọng số của cạnh khi chỉnh sửa
def update_edge_weight(i, j, widget):
    try:
        value = widget.get()
        if value == "":
            graph[i][j] = INF
        else:
            weight = int(value)
            if weight < 0:
                raise ValueError("Trọng số phải >= 0.")
            graph[i][j] = weight
    except ValueError:
        widget.delete(0, tk.END)
        widget.insert(0, "" if graph[i][j] == INF else
int(graph[i][j]))
    update_floyd_warshall()

# Hàm cập nhật ma trận Floyd-Warshall
def update_floyd_warshall():
    global dist, next_node
    dist, next_node = floyd_warshall(graph)
    draw_graph(canvas_frame, graph)

# Hàm vẽ đồ thị
def draw_graph(canvas_frame, graph, shortest_path=None):
    G = nx.DiGraph()
    n = len(graph)

    # Thêm các cạnh và trọng số vào đồ thị
    for i in range(n):
        for j in range(n):
            if graph[i][j] != INF and i != j:
```

```
                G.add_edge(i, j, weight=graph[i][j])

    pos = nx.spring_layout(G)  # Tính toán vị trí các đỉnh
    edge_labels = nx.get_edge_attributes(G, 'weight')

    # Tạo đồ thị với Matplotlib
    fig, ax = plt.subplots(figsize=(5, 5))
    nx.draw(G, pos, with_labels=True,
node_color='lightblue', node_size=3000, font_size=10,
ax=ax)
    nx.draw_networkx_edge_labels(G, pos,
edge_labels=edge_labels, ax=ax)

    # Vẽ đường đi ngắn nhất (nếu có)
    if shortest_path:
        edges = [(shortest_path[i], shortest_path[i + 1])
for i in range(len(shortest_path) - 1)]
        nx.draw_networkx_edges(G, pos, edgelist=edges,
edge_color='red', width=2, ax=ax)

    # Hiển thị đồ thị trên Tkinter Canvas
    for widget in canvas_frame.winfo_children():
        widget.destroy()  # Xóa đồ thị cũ
    canvas = FigureCanvasTkAgg(fig, master=canvas_frame)
    canvas.draw()
    canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

# Hàm cập nhật đồ thị khi thay đổi điểm bắt đầu và kết
thúc
def update_graph():
    try:
        start = int(start_entry.get())
        end = int(end_entry.get())
        if start < 0 or start >= len(graph) or end < 0 or
end >= len(graph):
            raise ValueError("Điểm bắt đầu hoặc kết thúc
không hợp lệ.")
    except ValueError:
        result_label.config(text="Vui lòng nhập số hợp lệ
(0-{}).".format(len(graph) - 1))
        return

    # Tìm đường đi ngắn nhất
    shortest_path = reconstruct_path(next_node, start, end)
    if shortest_path:
```

```
        distance = dist[start][end]
        result_label.config(text=f"Đường đi ngắn nhất:
{shortest_path}\nKhoảng cách: {distance}")
    else:
        result_label.config(text=f"Không có đường đi từ
{start} đến {end}.")

    # Vẽ đồ thị mới
    draw_graph(canvas_frame, graph, shortest_path)

# Tạo giao diện Tkinter
root = tk.Tk()
root.title("Giải thuật Floyd-Warshall")
root.geometry("1000x800")

# Khung chứa đồ thị
canvas_frame = tk.Frame(root)
canvas_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

# Khung chứa ma trận trọng số
matrix_frame = tk.Frame(root)
matrix_frame.pack(side=tk.TOP, padx=10, pady=10)

# Khung điều khiển
control_frame = tk.Frame(root)
control_frame.pack(side=tk.BOTTOM, fill=tk.X, padx=10,
pady=10)

# Điều khiển số đỉnh
tk.Label(control_frame, text="Số đỉnh:").pack(side=tk.LEFT)
vertex_entry = ttk.Entry(control_frame, width=5)
vertex_entry.insert(0, "5")
vertex_entry.pack(side=tk.LEFT, padx=5)

vertex_button = ttk.Button(control_frame, text="Tạo đồ
thị", command=update_graph_vertices)
vertex_button.pack(side=tk.LEFT, padx=10)

# Điều khiển điểm bắt đầu và kết thúc
tk.Label(control_frame, text="Điểm bắt
đầu:").pack(side=tk.LEFT)
start_entry = ttk.Entry(control_frame, width=5)
start_entry.insert(0, "0")
start_entry.pack(side=tk.LEFT, padx=5)
```

```
tk.Label(control_frame, text="Điểm kết
thúc:").pack(side=tk.LEFT)
end_entry = ttk.Entry(control_frame, width=5)
end_entry.insert(0, "4")
end_entry.pack(side=tk.LEFT, padx=5)

update_button = ttk.Button(control_frame, text="Cập nhật",
command=update_graph)
update_button.pack(side=tk.LEFT, padx=10)

# Nhãn kết quả
result_label = tk.Label(control_frame, text="")
result_label.pack(side=tk.LEFT, padx=10)

# Khởi tạo đồ thị mẫu
graph = create_sample_graph(5)
dist, next_node = floyd_warshall(graph)

# Hiển thị ma trận trọng số
show_weight_matrix(graph)

# Vẽ đồ thị ban đầu
draw_graph(canvas_frame, graph)

# Chạy ứng dụng
root.mainloop()
```