***UCS Algorithm***

import heapq

import matplotlib.pyplot as plt

import networkx as nx

def uniform\_cost\_search(graph, start, goal):

    """

    Perform Uniform Cost Search (UCS) on a graph.

    Args:

        graph (dict): The graph represented as an adjacency list where each node points to a list

                      of tuples (neighbor, cost).

        start (str): The starting node.

        goal (str): The goal node.

    Returns:

        path (list): The least-cost path from start to goal.

        total\_cost (float): The total cost of the path.

    """

    # Priority queue to store (cost, node, path)

    priority\_queue = [(0, start, [])]

    visited = set()

    while priority\_queue:

        # Pop the node with the smallest cost

        current\_cost, current\_node, path = heapq.heappop(priority\_queue)

        # If the node has already been visited, skip it

        if current\_node in visited:

            continue

        # Update path and mark node as visited

        path = path + [current\_node]

        visited.add(current\_node)

        # If the goal is reached, return the path and cost

        if current\_node == goal:

            return path, current\_cost

        # Add neighbors to the priority queue

        for neighbor, cost in graph.get(current\_node, []):

            if neighbor not in visited:

                heapq.heappush(priority\_queue, (current\_cost + cost, neighbor, path))

    # If the goal is not reachable, return failure

    return None, float('inf')

def visualize\_graph(graph, path=None):

    """

    Visualize the graph using matplotlib and networkx.

    Args:

        graph (dict): The graph represented as an adjacency list where each node points to a list

                      of tuples (neighbor, cost).

        path (list): The shortest path to highlight, if any.

    """

    G = nx.DiGraph()

    # Add edges to the graph

    for node, neighbors in graph.items():

        for neighbor, cost in neighbors:

            G.add\_edge(node, neighbor, weight=cost)

    pos = nx.spring\_layout(G, seed=42)  # Positioning for nodes (consistent with seed)

    edge\_labels = nx.get\_edge\_attributes(G, 'weight')

    # Draw nodes and edges

    nx.draw(G, pos, with\_labels=True, node\_size=500, node\_color="lightblue", font\_weight="bold", font\_size=10)

    nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels)

    # Highlight the path if provided

    if path:

        edges\_in\_path = [(path[i], path[i + 1]) for i in range(len(path) - 1)]

        nx.draw\_networkx\_edges(G, pos, edgelist=edges\_in\_path, edge\_color="red", width=2)

    plt.title("Graph Visualization with Shortest Path Highlighted")

    plt.show()

# Example usage

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

    # A more complex graph with additional nodes and edges

    graph = {

        'A': [('B', 1), ('C', 4), ('E', 7)],

        'B': [('A', 1), ('D', 2), ('F', 5)],

        'C': [('A', 4), ('D', 1), ('G', 3)],

        'D': [('B', 2), ('C', 1), ('H', 6)],

        'E': [('A', 7), ('F', 1), ('I', 8)],

        'F': [('B', 5), ('E', 1), ('J', 3)],

        'G': [('C', 3), ('H', 2), ('K', 4)],

        'H': [('D', 6), ('G', 2), ('L', 5)],

        'I': [('E', 8), ('J', 2), ('M', 7)],

        'J': [('F', 3), ('I', 2), ('N', 4)],

        'K': [('G', 4), ('L', 3)],

        'L': [('H', 5), ('K', 3), ('O', 6)],

        'M': [('I', 7), ('N', 1)],

        'N': [('J', 4), ('M', 1), ('O', 2)],

        'O': [('L', 6), ('N', 2)]

    }

    start\_node = 'A'

    goal\_node = 'O'

    # Run UCS and get the path and cost

    path, cost = uniform\_cost\_search(graph, start\_node, goal\_node)

    if path:

        print(f"Path: {' -> '.join(path)}, Cost: {cost}")

    else:

        print("No path found.")

    # Visualize the graph and highlight the path

    visualize\_graph(graph, path)

