**Single-Source Shortest Path Problem:**

def dijkstra(graph, source):  
     
    distances = {node: float('inf') for node in graph}  
    distances[source] = 0  
    previous = {node: None for node in graph}  
  
    unvisited\_nodes = list(graph.keys())  
  
    while unvisited\_nodes:  
        current\_node = min(unvisited\_nodes, key=lambda node: distances[node])  
  
        unvisited\_nodes.remove(current\_node)  
  
        for neighbor, weight in graph[current\_node].items():  
            distance = distances[current\_node] + weight  
  
            if distance < distances[neighbor]:  
                distances[neighbor] = distance  
                previous[neighbor] = current\_node  
  
    return distances, previous  
  
def main():  
    # Get the number of nodes from the user  
    num\_nodes = int(input("Enter the number of nodes: "))  
  
    # Get the graph from the user  
    graph = {}  
    for i in range(num\_nodes):  
        node = int(input(f"Enter node {i+1}: "))  
        neighbors = {}  
        num\_neighbors = int(input(f"Enter the number of neighbors for node {node}: "))  
        for j in range(num\_neighbors):  
            neighbor = int(input(f"Enter neighbor {j+1} for node {node}: "))  
            weight = int(input(f"Enter the weight for edge ({node}, {neighbor}): "))  
            neighbors[neighbor] = weight  
        graph[node] = neighbors  
  
    # Get the source node from the user  
    source = int(input("Enter the source node: "))  
  
    # Run Dijkstra's algorithm  
    distances, previous = dijkstra(graph, source)  
  
    # Print the shortest distances  
    print("Shortest Distances:")  
    for node, distance in distances.items():  
        print(f"{source} -> {node}: {distance}")  
  
    # Print the shortest paths  
    print("Shortest Paths:")  
    for node in graph:  
        if node != source:  
            path = []  
            current\_node = node  
            while current\_node is not None:  
                path.append(current\_node)  
                current\_node = previous[current\_node]  
            path.reverse()  
            print(f"{source} -> {node}: {' -> '.join(map(str, path))}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    main()

OUTPUT:

Enter the number of nodes: 5  
Enter node 1: 0  
Enter the number of neighbors for node 0: 2  
Enter neighbor 1 for node 0: 1  
Enter the weight for edge (0, 1): 5  
Enter neighbor 2 for node 0: 2  
Enter the weight for edge (0, 2): 7  
Enter node 2: 1  
Enter the number of neighbors for node 1: 2  
Enter neighbor 1 for node 1: 3  
Enter the weight for edge (1, 3): 10  
Enter neighbor 2 for node 1: 4  
Enter the weight for edge (1, 4): 2  
Enter node 3: 2  
Enter the number of neighbors for node 2: 1  
Enter neighbor 1 for node 2: 3  
Enter the weight for edge (2, 3): 6  
Enter node 4: 3  
Enter the number of neighbors for node 3: 1  
Enter neighbor 1 for node 3: 4  
Enter the weight for edge (3, 4): 10  
Enter node 5: 4  
Enter the number of neighbors for node 4: 3  
Enter neighbor 1 for node 4: 0  
Enter the weight for edge (4, 0): 12  
Enter neighbor 2 for node 4: 3  
Enter the weight for edge (4, 3): 5  
Enter neighbor 3 for node 4: 2  
Enter the weight for edge (4, 2): 18  
Enter the source node: 0  
Shortest Distances:  
0 -> 0: 0  
0 -> 1: 5  
0 -> 2: 7  
0 -> 3: 12  
0 -> 4: 7  
Shortest Paths:  
0 -> 1: 0 -> 1  
0 -> 2: 0 -> 2  
0 -> 3: 0 -> 1 -> 4 -> 3  
0 -> 4: 0 -> 1 -> 4