CNT Assignment 2

Name: Kuber Kishore **Class:** TY CSAI - A

Roll Number: 57 **Batch:** 3

Assignment: Write a program to find the shortest path using Dijkstra's Algorithm

Code

```
import heapq

def dijkstra(graph, start, end):
    # Distance from start to each node
    distances = {node: float('inf') for node in graph}
    distances[start] = 0

    # Previous node in optimal path
    previous = {node: None for node in graph}

    # Priority queue to get minimum distance node
    queue = [(0, start)]

    while queue:
        current_dist, current_node = heapq.heappop(queue)

        if current_node == end:
            break

        for neighbor, weight in graph[current_node]:
            distance = current_dist + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous[neighbor] = current_node
                heapq.heappush(queue, (distance, neighbor)) #relaxation

    # Reconstruct path
    path = []
    current = end
    while current:
        path.append(current)
        current = previous[current]
    path.reverse()

    return distances[end], path


def main():
```

```
    graph = {}
    n = int(input("Enter number of vertices: "))
    m = int(input("Enter number of edges: "))
    is_directed = input("Is the graph directed? (yes/no): ").strip().lower()
== 'yes'

    print("\nEnter all vertices:")
    vertices = []
    for i in range(n):
        v = input("Enter name for vertex " + str(i + 1) + ": ").strip()
        graph[v] = []
        vertices.append(v)

    print("\nEnter edges in format: from to weight")
    for _ in range(m):
        u, v, w = input().split()
        w = int(w)
        graph[u].append((v, w))
        if not is_directed:
            graph[v].append((u, w))

    start = input("\nEnter source vertex: ").strip()
    end = input("Enter destination vertex: ").strip()

    if start not in graph or end not in graph:
        print("Invalid vertices entered.")
        return

    distance, path = dijkstra(graph, start, end)

    if distance == float('inf'):
        print(f"No path found from {start} to {end}.")
    else:
        print(f"\nShortest path from {start} to {end}: {' -> '.join(path)}")
        print(f"Total distance: {distance}")


if __name__ == "__main__":
    main()
```

Output

Enter number of vertices: 6

Enter number of edges: 9

Is the graph directed? (yes/no): no

Enter all vertices:

Enter name for vertex 1: A

Enter name for vertex 2: B

Enter name for vertex 3: C

Enter name for vertex 4: D

Enter name for vertex 5: E

Enter name for vertex 6: F

Enter edges in format: from to weight

A B 1

A C 2

B D 3

B E 4

B C 3

C E 5

D E 3

D F 4

E F 1

Enter source vertex: A

Enter destination vertex: F

Shortest path from A to F: A -> B -> E -> F

Total distance: 6