```
class Graph:
    def _init_(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    def bellman_ford(self, src):
        dist = [float("inf")] * self.V
        dist[src] = 0
        prev_node = [-1] * self.V  # To store the previous node in the shortest path

        for _ in range(self.V - 1):
            for u, v, w in self.graph:
                if dist[u] != float("inf") and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w
                    prev_node[v] = u

        # Check for negative weight cycles
        for u, v, w in self.graph:
            if dist[u] != float("inf") and dist[u] + w < dist[v]:
                print("Graph contains negative weight cycle")
                return

        # Print distances from the source node
        print("Distances from the source node:")
        for i in range(self.V):
            print(f"Node {i}: {dist[i]}")
```

```
        # Print shortest paths from the source node to all other nodes
        for i in range(self.V):
            if i != src:
                self.print_shortest_path(src, i, prev_node, dist)

    def print_shortest_path(self, src, dest, prev_node, dist):
        path = [dest]
        while prev_node[dest] != -1:
            dest = prev_node[dest]
            path.insert(0, dest)

        print(f"\nShortest path from {src} to {path[-1]}:")
        print(" -> ".join(map(str, path)))
        print(f"Total distance: {dist[path[-1]]}")


def main():
    num_nodes = int(input("Enter the number of nodes: "))
    g = Graph(num_nodes)

    num_edges = int(input("Enter the number of edges: "))
    for _ in range(num_edges):
        edge_info = input("Enter edge (u v w): ").split()
        u, v, w = map(int, edge_info)
        g.add_edge(u, v, w)

    src_node = int(input("Enter the source node: "))
    g.bellman_ford(src_node)
```

```
if _name_ == "_main_":
    main()
```