```
from collections import deque
import heapq
from profiler import profile

@profile
def dijkstra(G, orig, dest, weightLabel='length', plot=False):

    for node in G.nodes:
        G.nodes[node]["visited"] = False
        G.nodes[node]["distance"] = float("inf")
        G.nodes[node]["previous"] = None
        G.nodes[node]["size"] = 0

    G.nodes[orig]["distance"] = 0
    G.nodes[orig]["size"] = 50
    G.nodes[dest]["size"] = 50
    pq = [(0, orig)]
    step = 0
    while pq:
        _, node = heapq.heappop(pq)
        if node == dest:
            break
        G.nodes[node]["visited"] = True
        for edge in G.out_edges(node):
            neighbor = edge[1]
            weight = G.edges[(edge[0], edge[1], 0)][weightLabel]

            if G.nodes[neighbor]["distance"] > G.nodes[node]["distance"] +
                G.nodes[neighbor]["distance"] = G.nodes[node]["distance"]
                G.nodes[neighbor]["previous"] = node
                heapq.heappush(pq, (G.nodes[neighbor]["distance"], neighb
        step += 1

    if G.nodes[dest]["previous"] is None:
        return None, step

    path = deque()
    current_node = dest
    while current_node is not None:
        path.appendleft(current_node)
        current_node = G.nodes[current_node]["previous"]
    return list(path), step


def dijkstra_end_node(G, start):
    # Initialize priority queue, distances, and previous node records
```

```
queue = [(0, start)]
distances = {node: float('infinity') for node in G.nodes}
previous_nodes = {node: None for node in G.nodes}
distances[start] = 0

while queue:
    current_distance, current_node = heapq.heappop(queue)
    for neighbor in G.neighbors(current_node):
        edge_data = G.get_edge_data(current_node, neighbor, 0)
        edge_speed = G.get_edge_data(current_node, neighbor, 0).get('
        edge_length = edge_data.get('length', 0)
        candidate_distance = current_distance + edge_length / (edge_sp

        if candidate_distance < distances[neighbor]:
            distances[neighbor] = candidate_distance
            previous_nodes[neighbor] = current_node
            heapq.heappush(queue, (candidate_distance, neighbor))

# Find the farthest node from the start node
furthest_node = max(distances, key=distances.get)
if distances[furthest_node] == float('infinity'):
    # return None if the farthest node cannot be reached
    return None, []

# Reconstruct the path from start to the furthest node
path = [furthest_node]
while previous_nodes[furthest_node] is not None:
    furthest_node = previous_nodes[furthest_node]
    path.append(furthest_node)
path.reverse()

return path[-1], path
```