```
# Practical work 3 -> problem 2

import heapq


def backwards_dijkstra(graph, start, end):
    # I initialize the distance dictionary, the priority queue and previous vertex dictionary
    distance = {vertex: float('inf') for vertex in graph.parse_vertices()}
    distance[end] = 0
    queue = [(end, 0)]  # the second element of the tuple is the distance, for the priority we take the lower
distance
    next = {vertex: None for vertex in graph.parse_vertices()}

    # I loop until I processed all the edges in the priority queue
    while queue:
        # Pop from priority queue the vertex with the lowest cost
        current_vertex, current_distance = heapq.heappop(queue)
        # I go to the next iteration if we have already a shorter path to the current vertex
        if current_distance != distance[current_vertex]:
            continue
        # I loop every neighbour of the current vertex and update the distance if needed
        for neighbor in graph.parse_inbound_edges(current_vertex):
            new_distance = current_distance + graph.get_cost((neighbor, current_vertex))
            if new_distance < distance[neighbor]:
                distance[neighbor] = new_distance
                heapq.heappush(queue, (neighbor, new_distance))
                next[neighbor] = current_vertex

    return distance[start], next


# (in main menu) ...

     # ...

          elif option == 17:  # shortest path
               start = int(input("Enter the start vertex: ").strip())
               if not graph.is_vertex(start):
                    raise ValueError(f"The vertex {start} does not belong to the graph!")
               end = int(input("Enter the end vertex: ").strip())
               if not graph.is_vertex(end):
                    raise ValueError(f"The vertex {end} does not belong to the graph!")
               dist, next = backwards_dijkstra(graph, start, end)
               path = []
               if next[start] is None:
                    print("There is no path between the two vertices!")
               else:
                    print(f"The shortest length from {start} to {end} is: {dist}")
                    current_vertex = start
                    # Here I construct the lowest cost path from the start vertex to the end vertex and I print it
                    while current_vertex != end:
                         path.append(current_vertex)
                         current_vertex = next[current_vertex]
                    path.append(end)
                    print(path)
```