```
infinit = float('inf') #infinity

# function for calculating the node with the shortest distance
def min_distance_node(distance, visited):
    min_dist = float('inf')
    min_dist_node = None
    for node in distance:
        dist = distance[node]
        if dist< min_dist and node not in visited:
            min_dist = dist
            min_dist_node = node
    return min_dist_node

def dijkstra(graph, start, target):
    # create a distance dictionary for all other nodes with distances set to infinity
    # starting distance set to 0
    distance = {node: infinit for node in graph}
    distance[start] = 0
    # create a dictionary to keep track of the parent nodes
    # all nodes set to None
    parents = {node: None for node in graph}
    # list of nodes that we have visited
    visited = []

    # select the current node as the lowest distance node that has not been visited yet
    current = min_distance_node(distance, visited)
    while current: # if all nodes have already been visited, finish the loop
        dist = distance[current] # get the distance to the current node
        neighbors = graph[current] # get the neighbors of the current node
        for n in neighbors.keys(): # Go through the neighbors of the current node
            new_dist = dist + neighbors[n] # calculate the new distance to that node
            if distance[n] > new_dist: # if it is shorter to get to the neighbor
                # by going through this node
                distance[n] = new_dist # update the distance for this node
                parents[n] = current # the node becomes the new parent of the neighbor
            visited.append(current) # mark the node as visited
        current = min_distance_node(distance, visited) # find the next node that we have to process
        # and continue looping through


    ## Recreate the path and the distance to the target node
    path = [] # list to store the nodes that we visited along the path
    if distance[target] is not infinit: # if the target node is reachable from the starting node
        current = target # start from the end
        # backtrack to recreate the path
        while current: # while current is not none
            path.append(current) # add the current node that we are at to the path
            current = parents[current] # go to the current node's parent node
            # then continue looping until there are no more parent nodes
        path.reverse() # reverse the order of the path so that we start at the starting node

    # returns a tuple with the path and the distance of the path
    return path, distance[target]
```