```
# This function will generate the path from source vertex to destination vertex
#   This function is called only if dist[destination] != infinity, which means that there is for sure a path.
def generatePath(prev, src, destination):
    """
    :param prev:  The list of predecessors for every vertex in the graph.
    :param src: The source vertex
    :param destination: the destination vertex
    :return: the path from src to destination
    """
    path = [destination]
    vertex = destination

    # We will parse backwards the list and after we reached the source vertex, we reverse the list
    # in order to have the list from source to destination
    while vertex != src:
        path.append(prev[vertex])
        vertex = prev[vertex]

    path.reverse()

    return path

def BellmanFordAlgortihm(graph, v1, v2):
    """
    This function will compute the minimum cost path from vertex v1 to vertex v2 if it exists
        - If in the graph there is a negative cost cycle, we throw an error.
    :param v2: the destination vertex
    :param v1: the source vertex
    :param graph: the graph object which contain all the vertices and all the edges.
    :return: the minimum cost of the walk from v1 to v2, or None if there is no walk from v1 to v2.
    """

    # We make all the distances equal to infinity
    dist = [float("inf")] * graph.NrOfVertices
    # Except for the source vertex
    dist[v1] = 0
    prev = [0] * graph.NrOfVertices
    changed = True
    while changed:
        changed = False
        for edge in graph.getAllEdges():
            vertex1 = edge[0]
            vertex2 = edge[1]
            cost = graph.getCostOfEdge(edge)
            if dist[vertex1] != float("inf") and dist[vertex2] > dist[vertex1] + cost:
                dist[vertex2] = dist[vertex1] + cost
                prev[vertex2] = vertex1
                changed = True

    # The above algorithm already produces the minimum costs, if there are no negative weight cycles.

    # In order to be sure that there are no negative weight cycles, we parse one more time through all the edges, and if
 there is
```

```
# edge for which the if statement is true, than there is a negative weight cycle.
for edge in graph.getAllEdges():
    vertex1 = edge[0]
    vertex2 = edge[1]
    cost = graph.getCostOfEdge(edge)
    if dist[vertex1] != float("inf") and dist[vertex2] > dist[vertex1] + cost:
        raise Exception("Negative cost cycle detected!\n")

# If dist[v2] is inf, it means that there is no path from v1 to v2.
if dist[v2] != float("inf"):

    path = generatePath(prev, v1, v2)
    return dist[v2], path

return None
```