Carp Nicoleta
Gr. 911

```
def Floyd_Warshall(g: GraphDirected):
    """
    1. We firstly initialize the distance matrix with 0 and the previous
matrix with "-"
    2. After we give the corresponding initial values by checking if the
created edge exists or not:
        if edge exists - dist will take the cost and prev the out vertex
        if not - the dist will take the established maximum value
    3. After we go through every possible path between two vertices and if
a path is a smaller cost appears we
       change it in the dist matrix

    :param g: a directed graph with non-negative costs
    :return: the prev and dist matrices, and the maximum value established
    """
    maxN = 10
    for i in g.edges.values():
        maxN += i

    dist = [[0 for i in range(g.nr_vert)] for j in range(g.nr_vert)]
    prev = [["-" for i in range(g.nr_vert)] for j in range(g.nr_vert)]

    for i in range(g.nr_vert):
        for j in range(g.nr_vert):
            if i != j:
                if g.is_edge(i, j):
                    dist[i][j] = g.edges[i, j]
                    prev[i][j] = i
                else:
                    dist[i][j] = maxN

    print("-- ORIGINAL --\nDistance: \n")
    print_matrix(dist, maxN)
    print("Previous: \n")
    print_matrix(prev, maxN)

    for k in range(g.nr_vert):
        print(f"\n k = {k}\n")
        for i in range(g.nr_vert):
            for j in range(g.nr_vert):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
                    prev[i][j] = prev[k][j]

        print("Distance: \n")
        print_matrix(dist, maxN)
        print("Previous: \n")
        print_matrix(prev, maxN)

    return dist, prev, maxN
```