SHORTEST PATH FROM ALL VERTICES TO ALL VERTICES

Using de Floyd-Warshall algorithm to compute the cost matrix, we will complete with a path matrix to get the list of vertices corresponding to the distances.

Complexity: O(nbVertices^3)

```
def to_matrix(self):
    """
    Function that creates two matrices of dimension n*n where n is the 
number of vertices.
    The indexes are the vertices of the graph.
    :return: dist - matrix where each cell dist[v1][v2] is filled with the 
cost corresponding to the edge (i,j)
                    if the edge doesn't exist, the value will be INF=9999
             path - matrix where each cell path[v1][v2] is filled with a 
list [i,j]
                    if the edge doesn't exist, the value will be an empty 
list []
    """
    INF=9999
    vertices=self.number_of_vertices()
    dist=[[INF for _ in range(vertices)] for _ in range(vertices)]
    path = [[[] for _ in range(vertices)] for _ in range(vertices)]
    for v in range(vertices):
        dist[v][v]=0
    for v1 in range(vertices):
        for v2 in range(vertices):
            if self.is_edge(v1,v2):
                dist[v1][v2]=self.get_cost(v1,v2)
                path[v1][v2]=[v1,v2]

    return dist,path
def floyd_warshall(self):
    """
    Floyd Warshall algorithm for computing the minimum cost distance and 
the paths between all vertices.

    :return:

    """
    dist,path=self.to_matrix()
    vertices = self.number_of_vertices()
    for k in range(vertices):
        for i in range(vertices):
            for j in range(vertices):
                if dist[i][j]>dist[i][k]+dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
                    path[i][j] = path[i][k] + path[k][j][1:]
        print(f"Intermediate matrix {k}: \n")
        for i in range(len(dist)):
            for j in range(len(dist[i])):
                print(dist[i][j], end=" ")
            print()
        print("\n")

    return dist,path
```

The algorithm starts with creating two matrices dist and path where:

> **dist** - matrix where each cell dist[v1][v2] is filled with the cost corresponding to the edge (i,j), if the edge doesn't exist, the value will be INF=9999
>
> **path** - matrix where each cell path[v1][v2] is filled with a list [i,j], if the edge doesn't exist, the value will be an empty list []

Every k iteration we create a new cost and path matrix. Let k be the intermediate vertex in the shortest path from source to destination.

In the first step, k is the first vertex. If the direct distance from the source to the destination is greater than the path through the vertex k, then the cell of dist is filled with dist[i][k] + dist[k][j] and the cell of path is filled with path[i][j] = path[i][k] + path[k][j][1:].

When k reaches a value equal to the number of vertices each cell will contain the minimum cost walk and the vertices of the shortest path between all vertices.