```python
def floyd_warshall(self, x, y):
    # check if the vertices are valid
    if not self.valid_vertex(x) or not self.valid_vertex(y):
        raise ValueError("Vertex not found")

    # define the infinity value
    infinity = 9999999

    # initialise the matrix of weights
    weights = [[infinity for i in range(self.__no_of_vertices)] for j in range(self.__no_of_vertices)]

    # initialise the path matrix
    paths = [[-1 for i in range(self.__no_of_vertices)] for j in range(self.__no_of_vertices)]

    # set the corresponding values in the weights and paths matrices
    for i in range(self.__no_of_vertices):
        for j in range(self.__no_of_vertices):
            if i == j:
                weights[i][j] = 0
            else:
                cost = self.check_edge(i, j)
                if cost is not None:
                    weights[i][j] = cost
                    paths[i][j] = i

    # use k as an intermediate vertex in order to find the shortest paths
    for k in range(self.__no_of_vertices):
        for i in range(self.__no_of_vertices):
            for j in range(self.__no_of_vertices):
                if weights[i][j] > weights[i][k] + weights[k][j]:
                    weights[i][j] = weights[i][k] + weights[k][j]
                    paths[i][j] = paths[k][j]
        print(f"\nINTERMEDIATE STATE with k = {k} as an intermediate vertex")
        print("\nWEIGHTS MATRIX:")
        self.print_matrix(weights)
        print("\nPATHS MATRIX:")
        self.print_matrix(paths)

    # check for negative cycles
    for i in range(len(weights)):
        if weights[i][i] < 0:
            print("We have a negative cycle")
            break

    # check if there exists a path from x to y
    if weights[x][y] == infinity:
        print(f"There is no path from {x} to {y}")
    else:
        # reconstruct the path
        p = [y]
        while p[-1] != x:
            if weights[p[-1]][p[-1]]:
                raise Exception("The path is not correct, it contains a negative cycle")
            p.append(paths[x][p[-1]])
        s = ""

        # format for printing
        aux = None
        for el in p[::-1]:
            if aux is not None:
                s += f"--{self.__dictionary_cost[(aux, el)]}>> {el} "
            else:
                s += f"{el} "
            aux = el
        print(f"path: {s}")
        print(f"cost: {weights[x][y]}")
```