# Practical work number 3

Function added to the ui class

```
def bellman_ford(self):
    try:
        source=int(input("What is the source vertex: "))
        dest=int(input("What is the destination vertex: ")) # we read from input
the vertices
        if source not in self.__graph.return_vertices() or dest not in
self.__graph.return_vertices():
            raise GraphException("Invalid inputs")
        distance,father=self.__graph.Bellman_Ford(source)
        path=[] # in this list will the path be retrieved
        current=dest # we go from finish to start
        while current!=-1: # we end the loop when we reach the source vertex(the
source vertex doesn't have a father)
            path.append(current)
            current=father[current] # the new current will be the father of the
previous current
        path.reverse()
        if distance[dest]==float("Inf"): # if there is no path, we signal that
            print("There is no path")
            return
        print("The distance between ",source," and ",dest," is:
",distance[dest])
        print("The path is: ",path)
    except GraphException as ge:
        print(ge)
```

The function added to the graph class

```
def Bellman_Ford(self,source):
    father=[-1 ]* self.__vertices # we set the fathers list -> everyone's father
is -1(nonexistent)
    distance=[float("Inf")]*self.__vertices # we set the distance to infinite to
each vertex
    distance[source]=0 # the distance to the source is set to 0
    for _ in range(self.__vertices-1): # we have at most nr_of_vertex iterations
        modification=False
        for start in self.__dicout.keys():
            for end in self.__dicout[start]:
                if distance[start]!=float("Inf") and
self.__costs[(start,end)]+distance[start]<distance[end]: # if we find a better
route to the end we update it
                    distance[end]=self.__costs[(start, end)] + distance[start] #
we update the distance to end
                    modification=True
                    father[end]=start # the new father/predecessor of end is
start
        if not modification: # if no modification was made in the current
iteration, we won't do another one, we return
            return distance,father
    for _ in range(self.__vertices-1): # we check if there is a negative cost
cycle, and if there is we throw an exception
        for start in self.__dicout.keys():
            for end in self.__dicout[start]:
                if distance[start]!=float("Inf") and
self.__costs[(start,end)]+distance[start]<distance[end]:
                    raise GraphException("There is a negative cost cycle")
    return distance,father # we return the distance and father lists
```