Pascuti Filip-Sergiu

Group 916/1

Documentation of the implementation

I have chosen Python for the first practical work.

```
class Directed_graph:
     def __init__(self, number_of_vertices):
        """
        Representation:
            - three dictionaries:
                    - one containing the inbound vertices of every
vertex (self._predecessors)
                    - one containing the outbound vertices of every
vertex (self._successors)
                    - one containing the cost of every edge
(self._edge_costs)
        """
     def get_edges_number(self):
        """
        method that returns the number of edges
        Input: -
        Output: the number of edges
        Exceptions: -
        """
     def get_vertices_number(self):
        """
        method that returns the number of vertices
        Input: -
        Output: the number of vertices
        Exceptions: -
        """
     def get_in_degree(self, vertex):
        """
        method that returns the in degree of a given vertex
        Input: vertex - the vertex for which we compute the in degree
        Output: the in degree of the wanted vertex (if the vertex does
 not exist, the output will be 0)
        """
```

```
    def get_out_degree(self, vertex):
        """
        method that returns the out degree of a given vertex
        Input: vertex - the vertex for which compute the out degree
        Output: the out degree of the wanted vertex
        """
    def get_edge_cost(self, x, y):
        """
        method that returns the cost of a given edge stores
        Input: x, y the source and target vertices of the edge
        Output: the cost of the edge (x, y)
        """


    def set_edge_cost(self, x, y, cost):
        """
        method that modifies the information of a given edge
        Input: x, y the source and target vertices of the edge, cost -
 the new cost of the edge
        Output: -
        """
    def iterate_vertices(self):
        """
        method that returns an iterator containing all the vertices
        Input: -
        Output: the iterator containing all the vertices
        """
    def iterate_outbound_edges(self, vertex):
        """
        method that returns an iterator containing the outbound edges
of a given vertex
        Input: vertex - the vertex for which we return the outbound
edges
        Output: the iterator containing the outbound edges
        """
    def iterate_inbound_edges(self, vertex):
        """
        method that returns an iterator containing the inbound edges
of a given vertex
        Input: vertex - the vertex for which we return the inbound
edges
        Output: the iterator containing the inbound edges
        """
```

```
    def is_edge(self, x, y):
        """
        method that checks if an edge exists
        Input: x, y - the source and target vertices
        Output: true - if the edge exists, false - if the edge does
not exist
        """
    def add_edge(self, x, y, cost=None):
        """
        method that adds an edge to the graph
        Input: x, y - the source and target vertices of the new edge,
cost - the cost of the edge
        Output: -
        Exceptions: ValueError exception if the edge already exists
        """




    def remove_edge(self, x, y):
        """
        method that removes an edge from the graph
        Input: x, y - the source and target vertices of the edge
        Output: -
        Exceptions: ValueError exception if the edge does not exist
        """
    def add_vertex(self, vertex):
        """
        method that adds a vertex to the graph
        Input: the vertex id
        Output: -
        Exceptions: ValueError exception if the vertex already exists
        """
    def remove_vertex(self, vertex):
        """
        method that removes a vertex from the graph along with its
connections
        Input: the vertex id
        Output:-
        Exceptions: ValueError exception if the vertex does not exist
        """
    def get_copy(self):
        """
        method that returns a deep copy of the graph
        Input: -
```

```
        Output: a deep copy of the graph
        Exceptions: -
        """

    def write_to_file(self, file_name):
        """
        method that writes the graph to a file
        Input: file_ame - the name of the file
        Output: -
        Exceptions: IO exception if something goes wrong with the
writing
        """
    def read_from_file(file_name):
    """
    method that reads a directed graph from a file and returns it
    Input: file_name - the name of the file
    Output: the graph that has been read from the file
    """




    def create_random_graph(number_of_vertices, number_of_edges):
    """
    method that creates a random graph with a given number of edges
and vertices, and writes it to a file
    Input:number_of_vertices - the number of vertices, number_of_edges
 - the number of edges, file_name - the name of the file the generated
 graph will be written
    Output: -
    Exceptions: ValueError exception if the number_of_edges >
number_of_vertices * (number_of_vertices - 1)
    """
```