**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)

    """