**Graph algorithms - practical work no. 1**

**Documentation – Bortoș Alexia-Ioana**

* **Specification:**

In order to design and implement an abstract data type *directed graph,* in Python, we shall define a class named *DirectedGraph,* providing the following properties and methods:

**@property  
def no\_of\_vertices(self):** *"""  
 :return: the number of vertices of the directed graph  
 """*

***@property  
def no\_of\_edges(self):*** *"""  
 :return: the number of edges of the directed graph  
 """*

***@property  
def vertices(self):*** *"""  
 :return: a set of all the vertices in the directed graph  
 """*

***@property  
def edges(self):*** *"""  
 :return: a set of all the edges in the directed graph  
 """*

***def get\_cost(self, source: int, target: int):*** *"""  
 Return the cost of an edge represented as (source, target)  
 :param source: the source of the edge  
 :param target: the target of the edge  
 :return: the cost of the edge  
 :raises: DirectedGraphException if either of the vertices does not exist or if their is no edge between source and target.  
 """*

***def set\_cost(self, source: int, target: int, cost: int):*** *"""  
 Return the cost of an edge represented as (source, target)  
 :param source: the source of the edge  
 :param target: the target of the edge  
 :param cost: the new cost of the edge  
 :return: None  
 :raises: DirectedGraphException if either of the vertices does not exist or if their is no edge between source and target.  
 """*

***def get\_in\_degree(self, vertex: int):*** *"""  
 Returns the in degree of the provided vertex. Raises DirectedGraphException if the vertex does not exist.  
 """*

***def get\_out\_degree(self, vertex: int):*** *"""  
 Returns the out degree of the provided vertex. Raises DirectedGraphException if the vertex does not exist.  
 """*

***def get\_inbound(self, vertex):*** *"""  
 Generator function, provides a way of iterating over the inbound edges of the provided vertex.  
 Raises DirectedGraphException if the provided vertex does not exist.  
 """*

***def get\_outbound(self, vertex):*** *"""  
 Generator function, provides a way of iterating over the outbound edges of the provided vertex.  
 Raises DirectedGraphException if the provided vertex does not exist.  
 """*

***def is\_vertex(self, vertex: int):*** *"""  
 Checks if a vertex exists in the graph or not.  
 :return: - True if the vertex exists  
 - False otherwise  
 """*

***def is\_edge(self, source: int, target: int):*** *"""  
 Checks if there is an edge between source and target.  
 Raises DirectedGraphException if either of the vertices does not exist.  
 :param source: The source of the edge  
 :param target: The target of the edge  
 :return: - True if there is an edge between source and target  
 - False otherwise  
 """*

***def add\_vertex(self, vertex: int):*** *"""  
 Adds a vertex to the directed graph. Raises DirectedGraphException if the vertex already exists in the graph.  
 :param vertex: the vertex to be added  
 :return: None  
 """*

***def remove\_vertex(self, vertex: int):*** *"""  
 Removes a vertex from the directed graph. Raises DirectedGraphException if the vertex does not exist.  
 :param vertex: the vertex to be removed  
 :return: None  
 """*

***def add\_edge(self, source: int, target: int, cost=None):*** *"""  
 Adds an edge to the directed graph.  
 Raises DirectedGraphException if either of the vertices does not exist or if the edge already exists.  
 :param source: the source of the edge  
 :param target: the target of the edge  
 :param cost: the cost of the edge  
 :return: None  
 """*

***def remove\_edge(self, source: int, target: int):*** *"""  
 Removes an edge from the directed graph.  
 Raises DirectedGraphException if either of the vertices does not exist or if the edge to be removed does not exist.  
 :param source: the source of the edge  
 :param target: the target of the edge  
 :return: None  
 """*

***def get\_copy(self):*** *"""  
 :return: a deep copy of the graph, it does not allow modifying the initial graph through itself.  
 """*

***def \_\_str\_\_(self):*** *"""  
 :return: A string representation of the directed graph.  
 """*

* **Implementation**

The class *DirectedGraph* has the following private data members:

***self.\_inbound\_edges = {}***

- dictionary used for storing the inbound edges of the graph’s vertices. The keys are the target vertices, the values are lists containing the source vertices corresponding to each key.  
***self.\_\_outbound\_edges = {}***

- dictionary used for storing the outbound edges of the graph’s vertices. The keys are the source vertices, the values are lists containing the target vertices corresponding to each key.

***self.\_\_no\_of\_vertices***

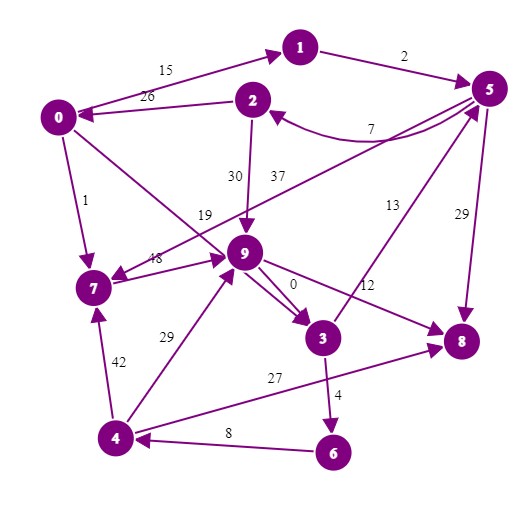
- the number of vertices of the graph, default value is 0 if none is provided at declaration  
***self.\_\_no\_of\_edges = 0***

- the number of edges of the graph  
***self.\_\_edges\_costs = {}***

- dictionary used for storing the edges of the graph, along with their corresponding cost. The keys are tuples of the form (source, target) i.e. the edges, the values are the corresponding costs of the edges.

* Function for generating a random *directed graph:*

**def generate\_random\_graph(no\_of\_vertices: int, no\_of\_edges: int):**  
 *"""  
 Generates a random graph with the specifed number of vertices and of edges.  
 :param no\_of\_vertices:  
 :param no\_of\_edges:  
 :return: the random graph  
 """* if no\_of\_edges > no\_of\_vertices \*\* 2:  
 raise DirectedGraphException("Too many edges for the specified number of vertices!")  
 graph = DirectedGraph(no\_of\_vertices)  
 while no\_of\_edges > 0:  
 source, target, cost = randint(0, no\_of\_vertices - 1), randint(0, no\_of\_vertices - 1), randint(0, 50)  
 try:  
 graph.add\_edge(source, target, cost)  
 no\_of\_edges -= 1  
 except Exception:  
 pass  
 return graph

Randomly generated graph with 10 vertices and 20 edges: ****