**Specification**

**Practical work no.1**

We shall define a class named DirectedGraph representing a directed graph.

The class DirectedGraph will provide the following methods:

def get\_vertices(self)

Returns the number of vertices of the graph.

def get\_edges(self)

Returns the number of edges of the graph.

def set\_vertices(self, value)

Sets the number of vertices.

def set\_edges(self, value)

Sets the number of edges.

def get\_in\_edges(self)

Returns the in edges.

def get\_out\_edges(self)

Returns the out edges.

def get\_costs(self)

Returns the edges and the costs associated to those.

def set\_in\_edges(self, value)

Updates the content of the in edges list.

def append\_to\_key\_in(self, key, value)

Appends a new in edge to the list of already existing ones.

def append\_to\_key\_out(self, key, value)

Appends a new out edge to the list of already existing ones.

def append\_costs(self, dict)

Appends a new edge with an associated cost to the list of already existing ones.

def set\_out\_edges(self, value)

Updates the out edges.

def set\_costs(self, key, value)

Updates the costs and the edges.

**Implementation**

def \_\_init\_\_(self):

*'''*

*Function that creates the directed graph entity(it creates the dictionaries of its edges)*

*Input: -*

*Output: -*

*'''*

#the in edges

self.\_\_inEdges = {}

#the out edges

self.\_\_outEdges = {}

#the edges with their associated cost

self.\_\_costs = {}

#the number of vertices

self.\_\_vertices = 0

#the number of edges

self.\_\_edges = 0

The edges are represented as dictionaries.For the in edges the key is the first vertex and the value is a list of out vertices,for the out edges the key is the first vertex and the value is a list of in vertices and for the costs the key is a tuple containing the vertices and the value is the cost of the edge.

Moldovan Vasilica-Andreea

Group 915/1