LAB Work no.1 Documentation

Truță David Cristian

```
def getCost(self, source, target):
    """
    Returns the cost of the edge between source and target
    :param source: a vertex(int)
    :param target: a vertex(int)
    :return: the cost of the edge between source and target(int)
    Rasies Exception if there is no edge between source and target
    """
    if self.existingEdge(source, target) is False:
        raise EdgeError("There is no Edge between these vertices.")
    return self._edges[(source, target)]

def setCost(self, source, target, cost):
    """
    Sets the cost of the edge between source and target, if the edge exists.
    :param source: a vertex(int)
    :param target: a vertex(int)
    :return:-
    Rasies Exception if there is no edge between source and target
    """
    if self.existingEdge(source, target) is False:
        raise EdgeError("There is no Edge between these vertices.")
    self._edges[(source, target)] = cost
```

```
def inboundEdge(self, vertex):
    """
    Returns a generator that can be converted to list, which is iterable and contains all inbound edges of vertex.
    :param vertex: a vertex(int)
    :return: generator
    """
    for v in self._dIn[vertex]:
        yield v

def outboundEdge(self, vertex):
    """
    Returns a generator that can be converted to list, which is iterable and contains all outbound edges of vertex.
    :param vertex: a vertex(int)
    :return: generator
    """
    for v in self._dOut[vertex]:
        yield v
```

```
    def getInDegree(self, vertex):
        """
        Returns the in degree of a vertex.
        :param vertex: a vertex(int)
        :return: (int) In degree of vertex
        """
        return len(self._dIn[vertex])

    def getOutDegree(self, vertex):
        """
        Returns the out degree of a vertex.
        :param vertex: a vertex(int)
        :return: (int) Out degree of vertex
        """
        return len(self._dOut[vertex])
```

```
102
103     def existingVertex(self, vertex):
104         """
105         Returns whether a vertex exists or not.
106         :param vertex:a vertex(int)
107         :return: True if the vertex exists, False otherwise
108         """
109         if vertex in self._dIn.keys():
110             return True
111         return False
112
113     def addVertex(self, vertex):
114         """
115         Adds a vertex to the graph.
116         :param vertex: a vertex(int)
117         :return:-
118         Raises VertexError if the vertex already exists
119         """
120         if self.existingVertex(vertex) is True:
121             raise VertexError("This vertex already exists! ")
122         self._dIn[vertex] = []
123         self._dOut[vertex] = []
124
125     def removeVertex(self, vertex):
126         """
127         Removes a vertex to the graph.
128         :param vertex: a vertex(int)
129         :return:
130          Raises VertexError if the vertex doesn't exists
131         """
132         if self.existingVertex(vertex) is False:
133             raise VertexError("This vertex doesn't exist")
134         # Removing edges that go in V
135         for v in self._dOut[vertex]:
136             self._edges.pop((vertex, v))
137             self._dIn[v].remove(vertex)
138         # Removing V from 'out' lists of every vertex that is in 'in' list of vertex AND the edge between them
139         for v in self._dIn[vertex]:
140             self._edges.pop((v, vertex))
141             self._dOut[v].remove(vertex)
142         # Removing V from dict of vertices
143         self._dIn.pop(vertex)
144         self._dOut.pop(vertex)
```

```
    def __insertVertex(self, source, target):
        """
        Helping function, it inserts vertices of an edge in the dictionaries.
        :param source: a vertex(int)
        :param target: a vertex(int)
        :return:-
        """
        self._dOut[source].append(target)
        self._dIn[target].append(source)

    def existingEdge(self, source, target):
        """
        Returns whether an edge exists or not.
        :param source:  a vertex(int)
        :param target:  a vertex(int)
        :return: True if the edge exists, False otherwise
        """
        if source in self._dIn[target]:
            return True
        return False

    def addEdge(self, source, target, cost=None):
        """
        Adds an edge to the graph.
        :param source: a vertex(int)
        :param target:a vertex(int)
        :param cost: The cost of the edge(int)
        :return:-
        Raises EdgeError if the edge already exists or if the vertices do not exist.
        """
        if self.existingVertex(source) is False or self.existingVertex(target) is False:
            raise EdgeError("Vertices or vertex of edge don't exist.")
        if self.existingEdge(source, target) is True:
            raise EdgeError(
                "This edge already exists. " + str((source, target)) + " " + str(self._edges[(source, target)]))
        self.__insertVertex(source, target)
        self._edges[(source, target)] = cost

    def removeEdge(self, source, target):
        """
        Removes an edge from the graph.
        :param source:a vertex(int)
        :param target:a vertex(int)
        :return:-
        Raises EdgeError if the edge does not exists.
        """
        if self.existingEdge(source, target) is False:
            raise EdgeError("This edge doesn't exist")
        self._dOut[source].remove(target)
        self._dIn[target].remove(source)
        self._edges.pop((source, target))
```

```
    def copyOfGraph(self):
        """
        Creates and returns a deepcopy of the Graph.
        :return: an instance of the DirectedGraph Class
        """
        newG = DirectedGraph()
        for v in self._dIn.keys():
            newG.addVertex(v)
        for edge in self._edges.keys():
            newG.addEdge(edge[0], edge[1], self._edges[edge])
        return newG
```