DOCUMENTATION

Group 911/2

Bulat Jaclina-Iana

MENU.py

```
import random
from Graph import GraphException, Graph

def graph_file(file):
    try: # tries to read from a file
        f = open(file,"rt") # opens the file
        l = f.readline() # reads the line of the file
        p, r = l.split(maxsplit=1,sep=" ") # splits the read information
        g = Graph(int(p)) # g takes value of the graph
        for i in range(int(r)):
            l = f.readline() # reads line until the last line is reached
            x, y, cost = l.split(maxsplit=2, sep=" ") # x,y and cost are read
            g.add_edge(int(x), int(y), int(cost)) # adds an edge to the graph
        f.close() # closes the file
        return g # return the graph read from the file
    except FileNotFoundError as e: # if the file is not found
        print(e) # prints an exception


def write_graph_to_file(g,file):
    f = open(file, "wt") # writes the graph to a file, here we open it
    f.write(str(g.get_number_of_vertices()) + " " + str(g.get_number_of_edges()) + "\n") # writes the
string into the file
    for key in g.iterate_edge(): # through the lines
        f.write(str(key[0])+ " "+ str(key[1])+ " "+ str(key[2])+"\n") # it writes the string
    f.close() # closes the file


def random_graph(n, m):
    if n*n < m:
        print("The given graph with "+ str(n)+ " vertices and "+ str(m) + " edges cannot be build. Try
again.") # Generates a random graph, if it cannot be built this string is shown
        return Graph() # returns the graph if it couldn t be built
    else:
        g = Graph(m) # g takes value of the graph
        x = random.randrange(n) # x is a random number
        y = random.randrange(n) # y is a random number as well
        for i in range(m): # in the range it has
            while g.check_edge(x, y): # while the edge is available
                x = random.randrange(n) # x takes another value
                y = random.randrange(n) # y takes another value
            g.add_edge(x, y, random.randrange(99)) # adds the edge to the graph
        return g # returns graph


def menu():
    # this function just prints the menu
    print("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~")
    print("~ ~ ~          M E N U                              ~ ~ ~")
    print("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~")
    print("<<             OPTIONS:                                >>")
    print("<<         1  Print vertices                           >>")
    print("<<         2  Print edges                              >>")
    print("<<         3  Add a vertex                             >>")
    print("<<         4  Add an edge                              >>")
    print("<<         5  Remove vertex                            >>")
    print("<<         6  Remove an edge                           >>")
    print("<<         7  Print the number of vertices             >>")
    print("<<         8  Print the number of edges                >>")
    print("<<         9  Print the IN degree of a vertex          >>")
    print("<<        10 Print the OUT degree of a vertex          >>")
```

```
    print("<<        11 Look if a vertex exists                    >>")
    print("<<        12 Look if an edge exists                     >>")
    print("<<        13 Get the cost of an edge                    >>")
    print("<<        14 Set the cost of an edge                    >>")
    print("<<        15 Print the OUT edges of a vertex            >>")
    print("<<        16 Print the IN edges of a vertex             >>")
    print("<<        17 Read a graph from a text file              >>")
    print("<<        18 Write a graph from a text file             >>")
    print("<<        19 Create a random graph                      >>")
    print("<<        0 EXIT                                        >>")
    print("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~")


def start():
    # we initialize a graph
    graph = None
    # while it has nothing in it, we read from the file
    while graph is None:
        f = input(">> Input the name of the file: ")
        graph = graph_file(f)
    while True:
        menu()
        # here we read the option we want it to take
        option = input(">> OPTION: ")
        try:
            if option == "1":
                for v in graph.iterate_vertices():
                    print(str(v)+" ", end="" + "\n")
            elif option == "2":
                for e in graph.iterate_edge():
                    print("EDGE: [" + str(e[0]) + "," + str(e[1]) + "]" + ", COST: " + str(e[2]) + " ",
end="" + "\n")
            elif option == "3":
                x = input(">> Input the vertex you want to add: ")
                graph.add_vertex(int(x))
                print("<< VERTEX ADDED >>")
            elif option == "4":
                x = input("<< Input the START:")
                y = input("<< Input the FINISH:")
                c = input("<< Input the cost:")
                graph.add_edge(int(x), int(y), int(c))
                print("<< EDGE ADDED >>")
            elif option == "5":
                x = input(">> Input the vertex you want to remove: ")
                graph.remove_vertex(int(x))
                print("<< VERTEX WAS REMOVED FROM THE GRAPH >>")
            elif option == "6":
                x = input(">> Input the START vertex: ")
                y = input(">> Input the FINISH vertex> ")
                graph.remove_edge(int(x), int(y))
                print("<< EDGE REMOVED >>")
            elif option == "7":
                print(str(graph.get_number_of_vertices()))
            elif option == "8":
                print(str(graph.get_number_of_edges()))
            elif option == "9":
                x = input(">> Input the vertex for the IN degree:")
                print(str(graph.in_degree(int(x))))
            elif option == "10":
                x = input(">> Input the vertex for the OUT degree: ")
                print(str(graph.out_degree(int(x))))
            elif option == "11":
                x = input(">> Input the vertex you want to see:")
                print(str(graph.check_vertex(int(x))))
            elif option == "12":
                x = input(">> Input the START vertex:")
                y = input(">> Input the FINISH vertex:")
                print(str(graph.check_edge(int(x), int(y))))
            elif option == "13":
                x = input(">> Input START:")
                y = input(">> Input FINISH:")
                print("COST: " + str(graph.get_edge_cost(int(x), int(y))))
            elif option == "14":
```

```
                x = input(">> Input START:")
                y = input(">> Input FINISH:")
                c = input(">> Input COST:")
                graph.set_edge_cost(int(x), int(y), int(c))
                print("<< COST HAS BEEN CHANGED >>")
            elif option == "15":
                x = input(">> Input vertex:")
                for vertex in graph.iterate_out(int(x)):
                    print(str(vertex) + " \n", end="")
            elif option == "16":
                x = input(">> Input vertex:")
                for vertex in graph.iterate_in(int(x)):
                    print(str(vertex) + " \n", end="")
            elif option == "17":
                graph = None
                while graph is None:
                    f = input(">> Input the file name: ")
                    graph = graph_file(f)
                print("<< SUCCESFULLY READ >>")
            elif option == "18":
                f = input(">> Input the file name: ")
                write_graph_to_file(graph, f)
                print("<< SUCCESFULLY WRITTEN >>")
            elif option == "19":
                g1 = random_graph(7, 20)
                g2 = random_graph(6, 40)
                write_graph_to_file(g1, "random_graph1.txt")
                write_graph_to_file(g2, "random_graph2.txt")
                print("<< GRAPHS CREATED SUCCESSFULLY >>")
            elif option == "0":
                return
            else:
                print("ERROR: NOT A VALID INPUT. Please try inputting something valid.")

        except (GraphException, ValueError) as r:
            print(r)

start()
```

Graph.py

```
import copy
class GraphException(Exception):
    pass

class Graph:
    def __init__(self, nr=0):
        """
        Constructor for the graph class
        :param nr: the number of vertices
        """
        self._out_vertices = dict()         #The dictionary of out vertices
        self._in_vertices = dict()          #The dictionary of in vertices
        self._cost = dict()                 #The dictionary of edge costs
        for i in range(0, nr): # we initialize them as lists
            self._out_vertices[i] = list()
            self._in_vertices[i] = list()

    def __copy__(self):
        # makes a deepcopy when it copies the graph
        return copy.deepcopy(self)

    def get_number_of_vertices(self):
        """
        Getter for the number of vertices
        :return: int(number of vertices)
        """
        return len(self._out_vertices)

    def get_number_of_edges(self):
        """
        Getter for the number of edges
```

```
        :return: int(number of edges)
        """
        return len(self._cost)

    def iterate_vertices(self):
        """
        Iterates through the vertices
        """
        for v in self._out_vertices:
            yield v

    def check_vertex(self, x):
        """
        Checks if there is a vertex in the graph
        :param x:
        """
        return x in self._out_vertices

    def check_edge(self, x, y):
        """
        Verifies if there is an edge between x and y
        :param x: the start point of the edge
        :param y: the end point of the edge
        :return: true, if the edge exists, false otherwise
        """
        return y in self._out_vertices[x]

    def in_degree(self, x):
        """
        getter for the IN DEGREE
        :param x: a vertex (we want to see the degree of)
        :return: The in degree of vertex x
        """
        if x not in self._out_vertices:
            raise GraphException("<< NO VERTEX FOUND >>")
        return len(self._in_vertices[x])

    def out_degree(self, x):
        """
        getter for the OUT DEGREE
        :param x: a vertex (we want to see the degree of)
        :return: The out degree of vertex x
        """
        if x not in self._out_vertices:
            raise GraphException("<< NO VERTEX FOUND >>")
        return len(self._out_vertices[x])

    def add_vertex(self, x):
        """
        Adds a vertex to the graph
        :param x: the vertex we want to be added
        """
        if x in self._out_vertices:
            raise GraphException("<< VERTEX ALREADY EXISTS >>")
        self._out_vertices[x] = list()
        self._in_vertices[x] = list()

    def iterate_out(self, x):
        """
        Iterates through the vertices x had an edge to
        :param x: the vertex
        """
        if x not in self._out_vertices:
            raise GraphException("<< NO VERTEX FOUND >>")
        for i in self._out_vertices[x]:
            yield i

    def iterate_in(self, x):
        """
        Iterates through the vertices that have an edge to x
        :param x: the vertex
        """
        if x not in self._out_vertices:
```

```
            raise GraphException("<< NO VERTEX FOUND >>")
        for i in self._in_vertices[x]:
            yield i

    def iterate_edge(self):
        """
        Iterates through the edges of the graph
        """
        for i in self._cost:
            yield i[0], i[1], self._cost[i]

    def add_edge(self, x, y, cost=0):
        """
        Adds an edge to the graph
        :param x: the out edge
        :param y: the in edge
        :param cost: the cost of the edge

        """
        if self.check_edge(x, y):
            raise GraphException("<< THE EDGE ALREADY EXISTS >>")
        self._out_vertices[x].append(y)
        self._in_vertices[y].append(x)
        self._cost[(x, y)] = cost

    def remove_edge(self, x, y):
        """
        Removes an edge from the graph
        :param x: in vertex
        :param y: out vertex
        """
        if not self.check_edge(x, y):
            raise GraphException("<< THE EDGE WAS NOT FOUND >>")
        self._out_vertices[x].remove(y)
        self._in_vertices[y].remove(x)
        self._cost.pop((x, y))

    def get_edge_cost(self, x, y):
        """
        Edge cost getter
        :param x: out vertex
        :param y: int vertex
        :return: the cost of the edge from vertex to vertex y
        """
        if not self.check_edge(x, y):
            raise GraphException("<< THE EDGE WAS NOT FOUND >>")
        return self._cost[(x, y)]

    def set_edge_cost(self, x, y, cost):
        """
        Edge cost setter, sets the cost of the edge from vertex to vertex y
        :param x: out vertex
        :param y: in vertex
        :param cost: new cost
        """
        if not self.check_edge(x, y):
            raise GraphException("<< EDGE WAS NOT FOUND >>")
        self._cost[(x, y)] = cost

    def remove_vertex(self, x):
        """
        Removes a vertex from the graph
        :param x: the vertex we want to remove from the graph
        """
        if x not in self._out_vertices:
            raise GraphException("<< NO VERTEX FOUND >")
        for i in self.iterate_vertices():
            if self.check_edge(x, i):
                self._in_vertices[i].remove(x)
                self._cost.pop((x, i))
            if self.check_edge(i, x):
                self._out_vertices[i].remove(x)
                self._cost.pop((i, x))
```

```
        self._out_vertices.pop(x)
        self._in_vertices.pop(x)
```