Documentation – Practical Work No 1

Groza Iulia – group 913

The Graph Class:

In the constructor, I use 3 dictionaries as follows:

1. self.inbound = {} – this is the dictionary for the predecessors, with vertices as keys, and the value is a list of all predecessors
2. self.outbound = {} – this is the dictionary for the successors, with vertices as keys, and the value is a list of all successors
3. self.costs = {} – this is the dictionary for the cost of the edges, with edges as keys, integer as the value

After the constructor, the class has another 14 functions:

1. add\_vertex
2. add\_edge
3. set\_cost
4. is\_edge
5. get\_cost
6. number\_of\_vertices
7. iterate\_set\_of\_vertices
8. in\_n\_out\_degree
9. iterate\_in

10. iterate\_out

11. remove\_vertex

12. remove\_edge

13. copy\_graph

14. reset

1. add\_vertex:

def add\_vertex(self, v):  
 *"""  
 this function adds the vertex as a key in both the vertices and edges dictionaries  
 and initializes their value with an empty list* ***:param*** *v: integer* ***:return****: raise ValueError if the vertex is already in the graph  
 """*

2. add\_edge:

def add\_edge(self, start, end, value):  
 *"""  
 checks if the vertices are part of the vertices and edges dictionaries, adds them if not and then adds the edge to the costs dictionary with the value assigned* ***:param*** *start: vertex* ***:param*** *end: vertex* ***:param*** *value: integer* ***:return****: raises ValueError if edge already exists  
 """*

3.set\_cost:

def set\_cost(self, start, end, value):  
 *"""  
 this function adds a value in the costs dictionary to a certain edge* ***:param*** *start: vertex* ***:param*** *end: vertex* ***:param*** *value: integer* ***:return****: raises ValueError if the edge doesn’t exist in the graph  
 """*

4. is\_edge:

def is\_edge(self, start, end):  
 *"""  
 this function checks if (start, end) is an existing edge in the graph* ***:param*** *start: vertex* ***:param*** *end: vertex* ***:return****: true if it is or false if it’s not  
 """*

5. get\_cost:

def get\_cost(self, start, end):  
 *"""*

*this function returns the cost of the edge (start, end)* ***:param*** *start: vertex* ***:param*** *end: vertex* ***:return****: integer  
 """*

6.number\_of\_vertices:

def number\_of\_vertices(self):  
 *"""* ***:return****: the number of vertices that the graph has  
 """*

7. iterate\_set\_of\_vertices

def iterate\_set\_of\_vertices(self):  
 *"""* ***:return****: a list of all the edges in the graph  
 """*

8.in\_n\_out\_degree:

*"""****:param*** *v: vertex****:return****: the in and out degree of vertex v, raises ValueError if vertex doesn't exist  
"""*

9.iterate\_in:

def iterate\_in(self, v):  
 *"""* ***:param*** *v: vertex* ***:return****: the inbound list of vertex v  
 """*

10.iterate\_out:

def iterate\_out(self, v):  
 *"""* ***:param*** *v: vertex* ***:return****: the outbound list of the vertex v  
 """*

11.remove\_vertex

def remove\_vertex(self, v):  
 *"""  
 this function removes a vertex from the inbound and outbound dictionaries*

*+ all the edges corresponding to that vertex from the cost dictionaries*

***:param*** *v: vertex* ***:return****: raises ValueError if the vertex doesn't exist  
 """*

12.remove\_edge

ef remove\_edge(self, start, end):  
 *"""  
 this function removes the edge (start,end) from all 3 dictionaries* ***:param*** *start: vertex* ***:param*** *end: vertex* ***:return****: raises ValueError if the edges is not in the graph*

13.copy\_graph

def copy\_graph(self):  
 *"""  
 this function creates another graph object with the same vertices, edges and costs of the edges as the current one* ***:return****: graph object  
 """*

14.reset

*"""  
this function resets all 3 dictionaries to be empty****:return****: nothing  
"""*

External functions:

I implement 6 external functions:

1. set\_cost\_to\_edge
2. random\_graph
3. read\_from\_file
4. write\_into\_file
5. valid
6. print\_options – function used to print the operations
7. menu – this is the function that displays the menu used to verify the operations

1.set\_cost\_to\_edge

*"""  
this function returns a random integer from 1 to 100 which can be assigned to an edge as its cost****:return****: integer  
"""*

2.random\_graph

def random\_graph(num\_v, num\_e):  
 *"""  
 this function builds a random generated graph* ***:param*** *num\_v: integer* ***:param*** *num\_e: integer* ***:return****: a graph object  
 """*

3.read\_from\_file

def read\_from\_file(file\_name):  
 *"""  
 Reads the information about a graph from a file, in the required form* ***:param*** *file\_name: a string, a valid file name* ***:return****: a graph object  
 """*

4.write\_into\_file

def write\_into\_file(graph, file\_name):  
 *"""  
 Writes the information about a graph in a file, in the required form* ***:param*** *graph: a graph object* ***:param*** *file\_name: a string, a valid file name* ***:return****: nothing  
 """*

5.valid

*"""  
Checks if a file\_name is valid <=> .txt appears in its name  
file\_name: a string****:return*** *true if it’s valid or false if not  
"""*