**Documentation for HW1 Graphs**

**(code was written in Python)**

General representation

We have 2 modules (1 for UI – user interface main\_ui.py that is testing the operations and some other functions for reading & writing to a file, and another one for Graph Representation graph\_representation.py which is a class that has a Triple Dictionary Graph Representation (as in the requirments): One dict for inbound edges/vertices, the second one for outbound edges/vertices, and the third one for storing as a key in a tuple (vertex1, vertex2) the value cost of each edge.

1. UI class is initialized with the following:
2. **Attributes**

**self.\_current\_index\_of\_graph\_list** – an element which memorizes the current graph with which the user is working it is set to None initially

**self.\_graphs** – a list which keeps the graphs created, used for switching between the graphs, being empty initially

1. **Methods**

**add\_empty\_graph(self)** – creates an empty graph and adds it to the list of graphs

available in the program, setting it to be the current graph also

**create\_random\_graph\_ui(self**) – takes as input from the user the number of vertices and edges and creates a random graph, adds it to the list and sets it to be the current graph

**generate\_random(self, vertices, edges)** – it is used by the create\_random\_graph\_ui function for generating a random graph; it raises a ValueError if the user provides too many edges and it returns the new graph

**read\_graph\_from\_file\_ui(self)** – takes the filename from the input of the user and it creates a new graph using the read\_graph\_from\_file, adds it to the list of graphs and sets it as the current graph

**write\_graph\_to\_file\_ui(self)** – writes using the function write\_graph\_to\_file the current graph that the program is working with in the format:

no\_of\_vertices no\_of\_edges

x y cost (this line can be repeated multiple times)

**switch\_graph\_ui(self)** – switch the graphs, the switch is done between available graphs in the self.\_graphs list

Precondition: the graph to be switched to has to exist in the list

& it also raises ValueError if the graph is not in the list

**get\_number\_of\_vertices\_ui(self)** – print the number of vertices the current graph has

**get\_number\_of\_edges\_ui(self)** – print the number of edges the current graph has

**list\_all\_outbound(self)** – lists all the vertices from the current graph with all their outbound vertices

**list\_outbound(self, x)** - lists the outbound vertices of the vertex x

**list\_all\_inbound(self)** – lists all the vertices from the current graph with all their inbound vertices

**list\_inbound(self, x)** - lists the inbound vertices of the vertex x

**list\_all\_costs(self)** – lists all the edges of the graph with their cost

**parse\_all\_vertices(self)** – list all the vertices of the graph

**add\_vertex\_ui(self)** – reads input from user for the vertex and uses the add\_vertex function to add a vertex to the graph; prints a corresponding message

**delete\_vertex\_ui(self)** – reads input from user for the vertex an uses the remove\_vertex function to remove a vertex from the graph; prints a corresponding message

**add\_edge\_ui(self)** – reads input for vertices and cost from the user and uses the add\_edge function to add an edge with a cost to the graph; prints a corresponding message

**remove\_edge\_ui(self)** - reads input for vertices from the user and uses the remove\_edge function to remove an edge from the graph; prints a corresponding message

**modify\_cost\_ui(self)** – read input from the user for the vertices and the cost and uses the change\_cost function to change the cost of an edge; prints a corresponding message

**get\_in\_degree\_ui(self)** – prints the in degree of a vertex given by the user; prints a corresponding message

**get\_out\_degree\_ui(self)** – prints the out degree of a vertex given by the user; prints a corresponding message

**check\_if\_edge\_ui(self)** - checks if an edge given by the user exists and if it does it prints its cost

**copy\_current\_graph\_ui(self)** – creates a copy of the graph using the make\_copy function

In the UI class we also have a function to print the menu, for the user to know its options and a starts the program. At the beginning the program initializes the current graph with an empty graph and gives the user various options to modify the elements. The graph can be stored in an output file and used at another run of the application.

1. Graph Representation Class:

Attributes/Data:

**self.\_number\_of\_vertices** – the number of vertices the graph has

**self.\_number\_of\_edges** – the number of edges the graph has

**self.\_dictionary\_in** – a dictionary for keeping the inbound vertices of each vertex, the vertices are the keys

**self.\_dictionary\_out** – a dictionary for keeping the outbound vertices of each vertex, the vertices are the keys

**self.\_dictionary\_cost** – a dictionary for keeping the edges and their costs, the edges are the keys

Methods:

dictionary\_cost(self)

Returns the dictionary of edges and costs of the graph

dictionary\_in(self)

Returns the dictionary of inbound vertices of the graph

dictionary\_out(self)

Returns the dictionary of outbound vertices of the graph

number\_of\_vertices(self)

Returns the number of vertices the graph has

number\_of\_edges(self)

Returns the number of edges the graph has

parse\_vertices(self)

Iterator for the vertices of the graph

parse\_inbound(self, x)

Iterator for the inbound vertices of the vertex x

parse\_outbound(self, x)

Iterator for the outbound vertices of the vertex x

parse\_cost(self)

Iterator for the edges of the graph and their costs

add\_vertex(self, x)

Add a new vertex x to the graph; returns False if it doesn’t add it and True otherwise Precondition: the vertex x must not already exist

remove\_vertex(self, x)

Remove a vertex x from the graph; return False if it doesn’t remove it and True otherwise

Precondition: the vertex x must be an existing one

in\_degree(self, x)

Return the in degree of the vertex x or -1 if the vertex does not exist Precondition: the vertex x must be an existing one

out\_degree(self, x)

Return the out degree of a vertex x or -1 if the vertex does not exist Precondition: the vertex x must be an existing one

add\_edge(self, x, y, cost)

Add a new edge (x, y) to the graph; returns False if it doesn’t add and True otherwise Precondition: the edge (x, y) must not already exist in the graph

remove\_edge(self, x, y)

Remove an edge (x, y) from the graph; returns False if it doesn’t remove it and True otherwise

Precondition: the edge (x, y) must already exist in the graph

find\_edge(self, x, y)

Returns the cost of the edge (x, y) if it exists and False otherwise

Precondition: the edge (x, y) must already exist in the graph

change\_cost(self, x, y, cost)

Change the cost of an edge (x, y) to be equal to cost; returns True if it changes, False otherwise

Precondition: the edge (x, y) must already exist in the graph

make\_copy(self)

Return a copy of the graph

**2 Additional functionalities for writing / reading in/from a file:**

**write\_graph\_to\_file(graph, file) -**Receiving as parameters a graph and a file it writes in a file the graph; it the file does

not exist it creates it; if the dictionaries used for printing, dictionary of costs and inbound dictionary, are empty a ValueError will be raised

Precondition: the graph shouldn’t be empty

**read\_graph\_from\_file(filename)**

It has as a parameter a file name from which the graph should be read

Precondition: the file must exist

<http://www.cs.ubbcluj.ro/~rlupsa/edu/grafe/lab1.html>

<http://www.cs.ubbcluj.ro/~rlupsa/edu/grafe/samples/graph.py>

https://www.w3schools.com/python/showpython.asp?filename=demo\_ref\_file\_readline