***Documentation***

**In graph.py**

**Graph – class:**

**\_\_init\_\_:**

Param – a list of nodes (that can be integers)

Creates graph\_out, graph\_in, edges dictionaries; the given nodes are put the first two dictionaries as keys, associated with empty lists initially.

For graph\_out and graph\_in for a given node a list is associated, that is the list of nodes which are connected to the so said node

For edges for a pair of nodes (start-point, end-point) an integer is associated, that is the cost of the edge between the two nodes

**nm\_of\_nodes:**

returns the number of element-pairs in the graph\_in dictionary (which is the number of nodes)

**nm\_of\_edges:**

returns the number of element-pairs in the edges dictionary

**is\_node:**

param - node

returns True if the node is in the graph\_in dictionary (therefore it is a node), False otherwise.

is\_edge

param – start, end

returns True is (start, end) tuple is in the edges dict; False otherwise

**add\_node:**

param – node

puts the given node into the graph\_in and graph\_out dictionary and associates it with an empty list.

Raises GraphException if is\_node(node) returns True

**remove\_node:**

param – node

removes the given node from the lists where it can be found (that is from graph\_in, graph\_out), deletes the elements from the edges dictionary, where the node is present as an element of the key, eventually deletes the key-list pairs from graph\_in, graph\_out, where the given node is a key.

Raises GraphException is is\_node(node) returns False

**add\_edge:**

param – start, end (these are nodes, integers), cost (integer)

puts into the edges dictionary (start, end) as keys and cost as value

appends to the graph\_out[start]’s list the node end, similarly, appends to the graph\_in[end]’s list the node start

raise GraphException if either of the given nodes are not nodes, or if the (start, end) is an edge

**remove\_edge:**

param- start, end

removes (start, end) from the dictionary edges, removes the element end from graph\_out[start]’s list, removes the element start from graph\_in[end]’s list

raises GraphException if either of the given nodes are not nodes, or if the (start, end) is not an edge

**get\_out\_degree:**

param – node

returns the length of the list graph\_out[node]

GraphException if node is not a node

**get\_in\_degree:**

param – node

returns the length of the list graph\_in[node]

GraphException if node is not a node

**iterate\_outbound\_edges\_of\_node:**

param – node

returns an iterator for the list of graph\_out[node]

GraphException if node is not a node

**iterate\_inbound\_edges\_of\_node:**

param – node

returns an iterator for the list of graph\_in[node]

GraphException if node is not a node

information\_of\_edge:

param – start, end

returns the value associat to (start, end) in the edges dictionary

GraphException if (start, end) is not an edge

modify\_edge:

param – start, end, cost

sets the value associated with (start, end) to cost in the edges dictionary

GraphException if (start, end) is not an edge

**GraphException – class:**

**\_\_init\_\_:**

param – message (a string)

Sets the property message to message

**\_\_str\_\_:**

Returns the property message

**In graph\_service.py**

**Service – class:**

**\_\_init\_\_:**

Param – graph (a Graph class object), text (string)

Sets the graph property to graph and the text property to text

**copy\_graph:**

returns a deepcopy of the graph property

get\_number\_of\_nodes:

See nm\_of\_nodes

**get\_nodes:**

returns a list of the graph\_in dictionary’s keys

**edge\_in\_between:**

param - start, end

returns True if is\_edge(start, end) or is\_edge(end, start) is True, False otherwise

**get\_edge\_cost:**

param – start, end

Calls the graph’s information\_of\_edge method

**set\_edge\_cost:**

param – start, end, cost

Calls the graph’s modify method

**get\_in\_degree:**

calls the graph’s get\_in\_degree method

**get\_out\_degree:**

calls the graph’s get\_out\_degree method

**iter\_outbound\_edges:**

param – node

calls and returns the graph’s iterate\_outbound\_edges\_of\_node

**iter\_inbound\_edges:**

param – node

calls and returns the graph’s iterate\_inbound\_edges\_of\_node

**load\_file:**

opens the text file name the text property reads and adds the edges found in the file to the graph

raises ServiceException if the file is not found or in other input output error cases

**write\_file:**

opens the text file name the text property, reads the number of nodes, number of edges and the edges from it and adds it to the graph

raises ServiceException if the file is not found or in other input output error cases

**add\_node:**

param – node

calls the graph’s add\_node method

**add\_edge:**

param – start, end, cost

calls the graph’s add\_edge method

**remove\_node:**

param – node

calls the graph’s remove\_node method

**remove\_edge:**

param – start, end

calls the graph’s remove\_edge method

**is\_edge:**

param – start, end

return the graph’s is\_edge method

**ServiceException – class:**

**\_\_init\_\_:**

param – message (a string)

Sets the property message to message

**\_\_str\_\_:**

Returns the property message

In **graph\_ui.py**

**class Console:**

\_\_init\_\_:

Param – service (object of Service class)

Sets the service property to service and creates a dictionary with pointers to the class’s method

**Read\_command:**

Prints out options asks for the users input and calls the corresponding method. Catches any exception raised and prints out the error.

**print\_options:**

Prints the options

a**dd\_node, add\_edge, remove\_node, remove\_edge, write\_to\_file, print\_node, get\_edge\_cost, set\_edge\_cost, degree\_of\_node, print\_outbound\_edge, print\_inbound\_edges –** all these methods asks for input from the user, calls the corresponding methods of the Service class and prints out the result.

**In main.py:**

**FileException – class:**

**\_\_init\_\_:**

param – message (a string)

Sets the property message to message

**\_\_str\_\_:**

Returns the property message

**generate\_random\_graph:**

param – n (number of nodes), m (number of edges)

returns a Graph class object with randomly generated nodes and edges (corresponding to the numbers)

**read\_graph\_from\_file:**

param – text (string)

creates a Graph object, opens the text file with the name text reads and adds the edges found in the file to the graph

raises FileException if the file is not found or in other input output error cases

Furthermore the main function asks from the user if he wants to generate the graph or read from the text, correspondingly asks for the number of nodes and edges or and calls the generate\_random\_graph function or the read\_graph\_from\_file function. Eventually creates a Service object, Console object and calls the Console object read\_command method.