**Practical work no. 1**

Specification

DirectedGraph class – represents a directed graph

* get\_vertices\_number(self) – returns the number of vertices of the current directed graph.
* get\_edges\_number(self) - returns the number of edges of the current directed graph.
* vertices\_iterator(self) – iterator for the vertices of the graph.
* is\_edge\_from\_first\_to\_second\_vertex(self, source\_vertex, destination\_vertex) – returns true if there is an edge from the vertex represented by the first parameter to the vertex represented by the second parameter.
* get\_in\_degree(self, vertex) – returns the in degree of the vertex provided.
* precondition: vertex provided must already exist in the graph.
* get\_out\_degree(self, vertex) – returns the out degree of the vertex provided.
* precondition: vertex provided must already exist in the graph.
* out\_vertices\_iterator(self, vertex) – iterator for the outbound vertices of the vertex provided.
* precondition: vertex provided must already exist in the graph.
* in\_vertices\_iterator(self, vertex) – iterator for the inbound vertices of the vertex provided.
* precondition: vertex provided must already exist in the graph.
* edges\_iterator(self) - iterator for the edges of the graph.
* get\_cost(self, edge) – returns the cost of the edge; throws exception if the edge does not exist.
* set\_cost(self, edge, new\_cost) – sets the cost of an edge with the value provided by the new\_cost parameter; throws exception if the edge does not exist.
* add\_edge(self, source\_vertex, destination\_vertex, cost=0) – adds new edge to the graph; throws exception if the vertices/edge does not exist.
* remove\_edge(self, source\_vertex, destination\_vertex) – removes the edge provided from the graph; throws exception if the vertices/edge does not exist.
* add\_vertex(self, vertex\_to\_be\_added) – adds a new vertex to the graph; throws exception if the vertex already exists.
* remove\_vertex(self, vertex\_to\_be\_removed) – removes a vertex from the current directed graph; throws exception if the vertex does not exist.
* create\_copy(self) – returns a copy of the current directed graph.

UI class – used for testing operations by user

* create\_empty\_directed\_graph(self) – creates an empty graph and adds it to the list of available directed graphs. It will become the currently working graph.
* create\_random\_directed\_graph(self) – takes from the user the number of vertices and edges that he wants and creates a random directed graph. It will become the currently working graph. It handles the case when there are passed more edges than the maximum possible number(there can not be generated such many edges with the given number of vertices).
* read\_directed\_graph\_from\_file\_ui(self) – takes the name of a file from the user as input and creates a new graph, adding it to the list of available graphs and setting it as the current graph.
* write\_directed\_graph\_to\_file\_ui(self) – writes the current graph to the file name provided by the user as input. It uses the write\_directed\_graph\_to\_file() method.
* switch\_to\_other\_directed\_graph(self) – switches the graphs between the available ones. The graph provided by the user as input has to exist in the list.
* display\_all\_outbound\_vertices(self) – prints all outbound vertices of the vertex provided by user as input.
* display\_all\_inbound\_vertices(self) - prints all inbound vertices of the vertex provided by user as input.
* get\_number\_of\_vertices\_ui(self) – prints the number of vertices of the current directed graph.
* get\_number\_of\_edges\_ui(self) – prints the number of edges of the current directed graph.
* get\_in\_degree\_of\_vertex(self) – prints the in degree of a vertex provided by the user. Prints the corresponding message.
* get\_out\_degree\_of\_vertex(self) - prints the out degree of a vertex provided by the user. Prints the corresponding message.
* display\_all\_edges\_and\_their\_costs(self) – prints all the edges of the directed graph and their corresponding costs.
* add\_vertex\_ui(self) - uses the add\_vertex() method to add an vertex provided as input by the user. Prints the corresponding message.
* remove\_vertex\_ui(self) - removes an vertex provided by the user from the graph using the remove\_vertex() method. Prints the corresponding message.
* add\_edge\_ui(self) – uses the add\_edge() method to add an edge provided as input by the user. Prints the corresponding message.
* remove\_edge\_ui(self) – removes an edge provided by the user from the graph using the remove\_edge() method. Prints the corresponding message.
* modify\_cost\_of\_edge(self) – uses the set\_cost() method to change the cost of an edge provided as input by the user. Prints a corresponding message.
* check\_if\_edge\_between\_vertices(self) – checks if an edge exists and prints its cost.
* parse\_vertices(self) – prints all vertices of the graph; if there are not any vertices, a suitable message will be printed.
* create\_copy\_of\_current\_graph(self) – creates a copy of the current directed graph using the create\_copy() method.
* print\_menu() – prints the menu for the user.

3 additional external functions

* read\_directed\_graph\_from\_file(file\_name) – reads a graph from the file provided as a parameter

-precondition: the file name provided must exist.

* write\_directed\_graph\_to\_file(directed\_graph, file\_name) – writes in a file the graph provided as a parameter; if the file does not already exist, it creates one.
* generate\_random\_graph(number\_of\_vertices, number\_of\_edges) – generates a random directed graph with the provided number of vertices and edges; it does not handle the case when there are passed more edges than the maximum possible number.

Implementation

DirectedGraph class

* self.\_\_vertices = list(vertex for vertex in range(vertices\_number))
* list containing the vertices of the directed graph.
* self.\_\_in\_vertices = {vertex: list() for vertex in self.\_\_vertices}
* dictionary, where the key represents all the vertices of the directed graph, and the value is a list containing all source vertices which form an edge with the key vertex, which is the destination vertex.
* self.\_\_out\_vertices = {vertex: list() for vertex in self.\_\_vertices}
* dictionary, where the key represents all the vertices of the directed graph, and the value is a list containing all destination vertices which form an edge with the key vertex, which is the source vertex.
* self.\_\_cost = dict()
* dictionary, where the key is a tuple representing an edge, formed by source and destination vertices, and the value represents the cost of that edge.

UI class

* self.\_\_available\_graphs = []
* list containing all the graphs that have been used since the start of the program.
* self.\_\_current\_graph = None
* an integer representing the index of the graph from the list on which the user is working.

DirectedGraph representation example

A diagram of a diagram

Description automatically generated with low confidence

self.\_\_vertices = (0, 1, 2, 3, 4)

self.\_\_in\_vertices = {0: [], 1: [3], 2: [1, 4], 3: [1, 4], 4: [0, 1]}

self.\_\_out\_vertices = {0: [1, 4], 1: [2, 3, 4], 2: [], 3: [], 4: [2]}

self.\_\_cost = {(0, 1): 3, (0, 4): 1, (1, 2): 1, (1, 3): 3, (1, 4): 1, (4, 2): 2}