Documentation

The following implementation contains a class defined Graph which represents the directed graph, another class UI which contains the corresponding functions for solving the problems and there are also 2 additional functions for reading a graph from a file and for writing a graph to a file.

The class Graph contains the following methods:

* get\_the\_number\_of\_vertices(self) – which returns the number of vertices of the graph
* get\_the\_number\_of\_edges(self) - which returns the number of edges of the graph
* get\_the\_dictionary\_for\_in(self) - which returns the dictionary of inbound vertices of the graph
* get\_the\_dictionary\_for\_out(self) - which returns the dictionary of outbound vertices of the graph
* get\_the\_dictionary\_costs(self) - which returns the dictionary of edges and costs of the graph
* parse\_vertices(self) – iterates the vertices of the graph
* add\_given\_edge(self, first, second, cost) - which adds a new edge (first, second) to the graph and returns False if it couldn’t be added and True otherwise. The edge (first, second) shouldn’t be in the graph.
* check\_if\_edge\_between\_vertices(self, first\_one, second\_one) – returns the cost of an edge if there is one between the given vertices or False otherwise
* parse\_edges\_outbound(self, given\_vertex) – iterates the outbound vertices of the given\_vertex
* parse\_edges\_inbound(self, given\_vertex) - iterates the inbound vertices of the given\_vertex
* get\_vertex\_in\_degree(self, given\_vertex) – returns False if the given\_vertex is not in the graph, or its in degree otherwise
* get\_vertex\_out\_degree(self, given\_vertex) - returns False if the given\_vertex is not in the graph, or its out degree otherwise
* modify\_the\_cost\_of\_given\_edge(self, first\_vertex, second\_vertex, new\_cost) – checks if the edge is in the graph and modify its cost if yes
* remove\_an\_edge(self, first, second) – removes an edge from the graph and return True if removed, False otherwise
* add\_vertex(self, given\_vertex) – adds a vertex to the graph, returns True if added and False if it is already in the graph
* remove\_vertex(self, given\_vertex\_to\_be\_removed) – removes a given vertex from the graph, returns True if it is removed and False if it not an existing one
* copy\_the\_graph(self) – creates and returns a copy of the current graph

The class Graph contains the following data:

* self.\_number\_of\_vertices – representing the number of vertices of the graph
* self.\_number\_of\_edges – representing the number of edges of the graph
* self.\_dictionary\_for\_in – representing a dictionary for the inbound vertices of each vertex, the keys being the vertices
* self.\_dictionary\_for\_out – representing a dictionary for the outbound vertices of each vertex, the keys are the vertices
* self.\_dictionary\_costs – representing a dictionary with the edges and their corresponding costs, the keys are the edges

The two additional functions for reading a graph from a file and writing the graph to a file are:

* read\_a\_graph\_from\_the\_file(file\_name) – takes the file\_name of an existing file and reads the graph from it
* write\_graph\_to\_file(graph, output\_file\_name) – it writes the graph into the given file, output\_file\_name, and raises an error is the graph couldn’t be written into the file

The UI class contains the following data being initialized:

* self.\_list\_of\_graphs which is a list containing the graphs are available from which the graphs can be switched
* self.\_current\_one – keep track of the current active graph or is None if no graph has been initialized

This class contains the following methods:

* initialize\_a\_new\_graph(self) – creates an empy graph and adds it to the list of the graphs that are available
* get\_number\_of\_vertices(self) – print the number of vertices of the current graph
* parse\_the\_set\_vertices\_ui(self) – list all the vertices
* find\_if\_there\_is\_edge\_between\_two\_given\_vertices(self) – checks if a given edge exists in the graph and prints its cost
* parse\_outbound\_edges(self) – list the outbound vertices of the given vertex
* parse\_inbound\_edges(self) – list the inbound vertices of the given vertex
* get\_the\_in\_degree\_of\_a\_vertex\_ui(self) – prints the in degree of the given vertex and the corresponding message otherwise
* get\_the\_out\_degree\_of\_a\_vertex\_ui(self) - prints the out degree of the given vertex and the corresponding message otherwise
* add\_given\_edge\_ui(self) – requests the corresponding input from the user and adds the edge and the given cost to the graph printing a corresponding message
* modify\_the\_costs\_ui(self) – modify the cost of the edge and prints a corresponding message if it was performed or not
* remove\_an\_edge\_ui(self) – removes an edge from the graph and prints a corresponding if the operation was done successfully or not
* add\_a\_vertex\_ui(self) – adds a vertex given by the user to the graph and prints its corresponding message
* remove\_a\_vertex\_ui(self) – removes a given vertex from the graph and prints the corresponding message
* copy\_the\_graph\_ui(self) – creates a copy of the current graph and adds it to the list of graphs
* read\_graph\_from\_file\_ui(self) – takes the file name given by the user and reads a graph from the given file and adds it to the list of graphs
* write\_graph\_to\_file\_ui(self) – writes the current active graph to the output file
* create\_a\_random\_graph\_ui(self) – creates a random graph after reading the input of the user for the number of vertices and edges
* generate\_randomly\_the\_graph(self, number\_of\_vertices, number\_of\_edges) – is used by create\_a\_random\_graph\_ui function to create randomly a graph and raises an error if the number of edges is too large
* show\_menu(self) – prints the menu with the available options for the user
* start(self) – this function keeps the program running until the user exits the program.