Documentation

The implementation uses the Python language.

For solving the problem statement we shall define:

- a class named **TripleDictGraph** which represents a directed graph,
- a class named **UI** for testing the operations
- and **4** additional **functions** for **reading** from a file and **writing** in a file.

In what follows, an **edge** is considered a **tuple (origin, target)**.

The isolated vertices are marked.

The class TripleDictGraph is initialized with the following 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

The class TripleDictGraph will provide the following 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

The 4 additional functions for reading/writing from/in a file are:

- write_graph_to_file(graph, file)
    - receiving as parameters a graph and a file it writes in a file the graph; if 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 and an error will be written in the file
- 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
- write_modified_graph_to_file(graph, file)
    - receiving as parameters a graph and a file it writes in a file the graph; if 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 and an error will be written in the file
- read_modified_graph_from_file(filename)
    - it has as a parameter a file name from which the graph should be read. Precondition: the file must exist

The UI class is initialized with the following data:

self._graphs

- a list that keeps the graphs created, used for switching between the graphs, it is empty initially

self._current

- an element that memorizes the current graph with which the user is working it is set to None initially

The UI class will provide the following 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)
    - the create_random_graph_ui function uses it 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 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 within the format: no_of_vertices no_of_edges x y cost (this line can be repeated multiple times)
- read_modified_graph_from_file_ui(self)
    - takes the filename from the input of the user and creates a new graph using the read_modified_graph_from_file, adds it to the list of graphs, and sets it as the current graph
- write_modified_graph_to_file_ui(self)
    - writes using the function write_modified_graph_to_file the current graph that the program is working within the format: 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 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
- 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 the 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 the user for the vertex and 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 start 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 application run.