# Documentation

For the implementation I am using the Python language. For representing the directed graph I shall define a class name TripleDictionaryGraph and for providing a menu for the user I shall define a class UI. In my representation, an edge is considered to be a tuple (origin, target).

The class TripleDictionaryGraph will provide the following methods:

• get\_number\_of\_edges(self)

Returns the number of edges of the graph

• get\_number\_of\_vertices(self)

Returns the number of vertices of the graph

• parse\_set\_vertices(self)

Iterator for the vertices of the graph

• check\_edge(self, vertex1, vertex2)

Checks if there is an edge between to given vertices returning the edge if it exists or returning false, otherwise

• get\_in\_degree(self, vertex)

Returns the inbound deegre of a given vertex if the vertex exists or returns -1 if it doesn’t exist

• get\_out\_degree(self, vertex)

Returns the outbound deegre of a given vertex if the vertex exists or returns -1 if it doesn’t exist

• parse\_outbound\_vertices(self, vertex)

Iterator for the outbound vertices of the graph, returns -1 if the vertex does not exist

• parse\_inbound\_vertices(self, vertex)

Iterator for the inbound vertices of the graph, returns -1 if the vertex does not exist

• change\_cost(self, edge\_x, edge\_y, cost)

Changes the cost of the edge (edge\_x, edge\_y) with the cost provided by the user and returns True if the edge exists, if it doesn’t,

it returns False

• add\_vertex(self, vertex)

Adds a new vertex to the graph and returns True if the operation was done successfully, returning False otherwise

• add\_edge(self, vertex1, vertex2, cost)

Adds a new edge to the graph with a given cost and returns True if the operation was successful. If one of the vertices does not exist or the edge (vertex1, vertex2) already exists it returns False and doesn’t perform the operation

• remove\_edge(self, vertex1, vertex2)

Removes the edge (vertex1, vertex2) and returns True if the operation was successful or returns False if the edge does not exist. The method also removes from the other vertices the outbound and inbound vertices

• remove\_vertex(self, vertex)

Removes the vertex given by the user from the graph and returns True if the operation was successful or returns False if the vertex does not exist. It also removes all the edges that include the vertex

• parse\_cost(self)

Iterator for the cost

• copy\_graph(self)

Returns the copy of the graph

• read\_from\_file(filename)

A static method that creates a graph from the information from the given filename. The file must exist

• write\_graph\_to\_file(graph, file)

A static method that writhes the given graph to a file. If the file does not exist the method creates it. If the graph is empty it will raise a ValueError

The class TripleDictionaryGraph is initialized with the following data:

self.number\_of\_verticies – the number of vertices the graph has

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

self.inbound\_dictionary – a dictionary where are stored the inbound vertices of each vertex, the vertices being the keys

self.outbound\_dictionary – a dictionary where are stored the outbound vertices of each vertex, the vertices being the keys

self.cost\_dictionary – a dictionary where are stored the costs of each edge, the edges (vertex\_x, vertex\_y) being the keys

The UI class is initialized with the following data:

self.\_\_graphs – a list where the created graphs are being stored, where the user can switch the graphs. It is initially empty

self.\_\_current\_graph – an index for iterating through the graphs. It is initially set to None

The UI class has the following methods:

add\_empty\_graph(self)

Adds an empty graph to the list of graphs and prints a message. It is used when starting the application.

generate\_random\_graph(vertices, edges)

It is a static method that generates and returns a random graph with the number of vertices and edges given by the user. If the number of edges provided is bigger than the number of vertices squared a ValueError will be raised.

ui\_create\_random\_graph(self)

Gets the input from the user to create a random graph and adds it to the list of graphs

ui\_read\_graph\_from\_file(self)

Reads the file from the filename provided by the user and adds it to the graphs list

ui\_write\_graph\_to\_file(self)

Writes the graph indicated by the self.\_\_curent\_graph to a file

ui\_switch\_graph(self)

Allows the user to switch to any graph from the graphs list

ui\_get\_number\_of\_vertices(self)

Lists the number of vertices

ui\_get\_number\_of\_edges (self)

Lists the number of edges

list\_all\_outbound(self)

Lists all the vertices with all their outbound vertices

list\_outbound(self)

Lists all the outbound vertices of the graph

list\_all\_inbound(self)

Lists all the vertices with all their inbound vertices

list\_inbound(self)

Lists all the inbound vertices of the graph

list\_all\_costs(self)

Lists all the edges and their costs

list\_all\_vertices(self)

Lists all the vertices

ui\_add\_vertex(self)

Gets the input from the user to add a new vertex

ui\_remove\_vertex(self)

Gets the input from the user to remove the vertex

ui\_add\_edge(self)

Gets the input from the user to add a new edge

ui\_remove\_edge(self)

Gets the input from the user to remove an edge

ui\_modify\_cost(self)

Gets the edge from the user to modify the cost of it

ui\_get\_in\_degree(self)

Gets the vertex from the user and lists its inbound degree

ui\_get\_out\_degree(self)

Gets the vertex from the user and lists its outbound degree

ui\_check\_edge(self)

Gets the edge from the user and prints if it is a edge or not

ui\_copy\_current\_graph(self)

Makes a copy of the current graph and adds it to the graphs list

print\_menu()

A static method that prints the menu with the options for the user

start(self)

Starts the application. It initializes an empty graph and provides the menu for the user waiting for its input. According to the input one of the UI methods is called and it keeps running until the user provides “x” or an exception its thrown.

DirectedGraph representation example

A diagram of a diagram

Description automatically generated with low confidence

self.number\_of\_vertices = 5

self.number\_of\_edges = 7

self.inbound\_dictionary = {0: [], 1: [3], 2: [1, 4], 3: [1, 4], 4: [0, 1]}

self.outbound\_dictionary = {0: [1, 4], 1: [2, 3, 4], 2: [], 3: [], 4: [2]}

self.cost\_dictionary = {(0, 1): 3, (0, 4): 1, (1, 2): 1, (1, 3): 3, (1, 4): 1, (4, 2): 2}