PRACTICAL WORK #1

Specification:

We shall define a class named OrientedGraph representing a directed graph.

The class OrientedGraph will provide the following methods:

- def __init__(self, filename):
  - constructs a graph using 3 dictionaries, dict_in represents the in edges, has as keys the vertices and as value a list containing the start vertice for an in edge. dict_out represents the out edges and works similary as dict_in. dict_costs represents the costs of every edge, has as keys tuples representing an edge and as values the costs for the edges. The data for the graph are in the file "filename".
- def add_edge(self, vertex, direction, cost):
  - adds an edge
- def search_edge(self, vertex, direction):
  - searches an edge
- def retrieve_info(self, vertex, direction):
  - retrieves the cost of an edge

- search_vertex(self, vertex):
  - searches ~~on~~ a vertex

- add_vertex(self, vertex):
  - adds a vertex

- def delete_edge(self, vertex, direction)
  - deletes an edge

- def remove_vertex(self, vertex):
  - removes a vertex

- def get_nr_of_vertices(self):
  - returns the nr of vertices from the graph

- def nr_of_edges(self):
  - returns the nr of edges from the graph

- def copy_graph(self):
  - makes a copy of the current graph and saves it in a file named "copy_graph"

Implementation

There are 2 private functions in the Oriented Graph that will load and save to a specific file the content of the graph.

- def - load_file(self):

  - loads the content from a file, representing a graph, in the memory.

- def - save_file(self):

  - saves to a file the content of the graph from the memory.

The class ServiceOrientedGraph works like a bridge from the UI to the OrientedGraph, such that there will not be a direct connection between the 2. All the methods from the service work the same as the ones from the OrientedGraph.