Practical work no. 1 – Documentation

Specifications

We shall define a class named **DirectedGraph** representing a directed graph.

The class **DirectedGraph** will provide the following methods:

- def __init__(self, number_of_vertices=0, number_of_edges=0):
  - The class constructor

- def __str__(self):
  - Returns the string representation of current graph

- def get_edges_costs(self):
  - Returns the dictionary (of edges and their cost) of the graph

- def get_inbound_vertices(self):
  - Returns the dictionary of the inbound vertices of the graph

- def get_outbound_vertices(self):
  - Returns the dictionary of the outbound vertices of the graph

- def get_number_of_vertices(self):
  - Returns the number of vertices for current graph

- def get_number_of_edges(self):
  - Returns the number of vertices for current graph

- def add_vertex(self, vertex_to_add):
  - Precondition: vertex must not already exist in graph
  - Add a new vertex to current graph
  - Returns: True (if vertex can be added) ; False (otherwise)

- def remove_vertex(self, vertex_to_remove):
  - Precondition: vertex must already exist in graph
  - Remove a vertex from current graph
  - Returns: True (if vertex can be removed) ; False (otherwise)

- def add_edge(self, vertex1, vertex2, cost):
  - Precondition: edge must not already exist in graph
  - Add an edge to current graph
  - Returns: True (if edge can be added) ; False (otherwise)

- def remove_edge(self, vertex1, vertex2):
  - Precondition: edge must already exist in graph
  - Remove an edge from current graph
  - Returns: True (if edge can be removed) ; False (otherwise)

- def update_edge_cost(self, vertex1, vertex2, cost):
  - Precondition: edge must already exist in graph
  - Updates the cost of the edge represented by the (start edge, destination_edge)

- def parse_vertices(self):
  - Iterator for the vertices of graph

- def parse_inbound_vertices(self, vertex):
  - Iterator for the inbound vertices of given vertex

- def parse_outbound_vertices(self, vertex):
  - Iterator for the outbound vertices of given vertex

- def get_in_degree(self, vertex):
  - Precondition: Vertex must exist
  - Returns the in degree of vertex or -1 if it doesn’t exist
- def get_out_degree(self, vertex):
  - Precondition: Vertex must exist
  - Returns the out degree of vertex or -1 if it doesn’t exist

- def make_copy_of_current_graph(self):
  - Makes a copy of current graph and and returns the last copy from the list of copies.

Implementation

The class **DirectedGraph** is initialized with the following data:

- self._nr_vertices = the number of vertices for current graph
- self._nr_edges = the number of edges for current graph
- self.inbound_vertices = a dictionary made for keeping the inbound vertices for each vertex
- self.outbound_vertices = a dictionary made for keeping the outbound vertices for each vertex

- self.edges_costs = a dictionary made for keeping track of each edge (the key) and its cost (the value).

  The **external functions** are:

- def generate_random_graph(num_vertices, num_edges):
  - Generates a random graph using the given number of vertices and edges
  - Returns the random graph

- def read_from_file(file_name):
  - Precondition: File must exist
  - It receives the file_name from which the current graph should be read

- def write_in_file(file_name,given_graph):
  - It receives as parameters the name of the file where it writes the graph and the graph that has to be written.