Practical work no.1

Documentation

-Turcu Ciprian-Stelian gr.917/2-

In the implementation of this program, using the Python language, we shall use a class named Graph which represents a directed graph, a class named Ui for the possibility of testing the operations and 2 additional functions for reading from a file and writing in a file. In the implementation an edge is considered as a tuple (x, y)/ (source, target).

The class Graph has the following methods:

- def dict_cost(self)
  Returns the dictionary of edges and costs of the directed graph.
- def vertices_in(self):
  Returns the dictionary of inbound vertices of the graph.
- def vertices_out(self):
  Returns the dictionary of outbound vertices of the graph.
- def number_vertices(self):
  Returns the number of vertices of the graph.
- def number_edges(self):
  Returns the number of edges in the graph.
- def parse_vertices(self):
  Method that gets an iterator for the vertices of the graph.
- def parse_cost(self):
  Method that gets an iterator for the graph and their costs.
- def parse_in(self, x):
  Iterator for the inbound vertices of the vertex x.
- def parse_out(self, x):
  Iterator for the outbound vertices of the vertex x.
- def add_vertex(self, x):
  Adds a new vertex x to the graph.
  Returns False if the addition wasn’t successful and True otherwise.
  Precondition: the vertex x must not already exist.
- def remove_vertex(self, x):
  Removes the vertex x from the graph.
  Returns False if the removal wasn’t successful and True otherwise.
  Precondition: the vertex x must be in the graph.

- def get_in_degree (self, x):
  Returns the in degree of the vertex x or False if the vertex does not exist.
  Precondition: the vertex x must be in the graph
- def get_out_degree (self, x):
  Returns the out degree of the vertex x or False if the vertex does not exist.
  Precondition: the vertex x must be in the graph.
- def add_edge (self, x, y, cost):
  Adds a new edge represented as 2 endpoints x and y as a tuple (x, y) to the graph and assigns a cost to the tuple.
  Returns False if the addition wasn't successful and True otherwise.
  Precondition: the vertex x and y must exist and the edge(x,y) must not be already in the graph.
- def remove_edge (self, x, y):
  Removes an edge represented as 2 endpoints x, y as a tuple (x, y) from the graph.
  Returns False if the removal wasn't successful and True otherwise.
  Precondition: x, y must be vertices within the graph and the edge (x, y) must exist in the graph.
- def check_edge (self, x, y):
  Returns the cost of a given edge represented as 2 endpoints x, y as a tuple (x, y) from the graph or False if it does not exist.
  Precondition: the edge(x,y) must already exist in the graph.
- def change_cost_edge (self, x, y, new_cost):
  Changes the cost of an edge represented as 2 endpoints x, y as a tuple (x, y) from the graph with the new given cost.
  Returns False if the change wasn't successful or True otherwise.
  Precondition: the edge (x, y) must be in the graph.
- def copy_data(self):
  Creates a copy of the graph and returns it.

The class Graph is initialized with the following data:

- self._number_vertices – the number of vertices in the graph
- self._number_edges – the number of edges the graph has
- self._vertices_in – a dictionary for keeping the inbound vertices of each vertex, the vertices are the keys
- self._vertices_out – a dictionary for keeping the outbound vertices of each vertex, the vertices are the keys
- self._dict_cost – a dictionary for keeping the edges and their costs, the edges are the keys

Functions for reading/writing from/in a file are:

- write_to_file(filename, graph)
  Receiving as parameters a file and a graph it writes in the specified file the graph; if the file does not exist it creates it. If the dictionaries used in writing are empty a ValueError will be raised.
  Precondition: the graph should not be empty.

The Ui class is initialized with the following data:

- self._graph_list – a list which keeps the graphs created, used for switching between the graphs.
- self._current – an element which memorizes the current graph whit which the user is working.

The class Ui has the following methods:

- get_nr_of_vertices(self) – print the number of vertices of the current graph.
- parse_vertices(self) – print all the vertices in a graph.
- check_if_edge(self) – prints to the user if a tuple (x, y) is an edge in the graph.
- get_in_degree_of_vertex(self) – prints for the user the in degree of a given vertex.
- get_out_degree_of_vertex(self) - prints for the user the out degree of a given vertex.
- parse_outbound_edges_of_vertex(self) – reads a vertex and prints the outbound edges of the vertex.
- parse_inbound_edges_of_vertex(self) – reads a vertex and prints the inbound edges of the vertex.
- modify_cost_edge(self) – reads an edge and a new cost and modifies the cost with the new one if the edge is in the graph.
- add_vertex_ui(self) – reads a vertex and adds it to the graph if it doesn't already exist.
- delete_vertex(self) -reads a vertex and deletes that vertex form the graph and the relation edges if the vertex is in the graph.
- add_edge_ui(self) – reads an edge and adds it to the graph if it is not already present.
- delete_edge(self) – reads an edge and deletes the edge from the graph if it is present in the graph.
- copy_graph(self) – makes a copy of the graph and adds it to the graph_list.
- read_graph_from_file(self) – reads a filename and reads the graph from the file.
- Write_graph_to_file(self) – reads a filename and writes the current graph into the destination file.
- ui_generate_graph_random(self) – reads a number of edges and vertices and creates a graph which is then added to the graph_list.

- generate_random_graph(self , v, e) – creates a random valued graph respecting the given boundaries v and e.