Moisuc Ana-Doris, group 915

Graph Algorithms- practical work no1

Documentation for the implementation in python

We have a new class: *Graph*

Fields:

- vertices– the number of vertices
- dictionaryIn – the dictionary containing for each vertex the list of inbound neighbours
- dictionaryOut – the dictionary containing for each vertex the list of outbound neighbours
- dictionaryCosts – the dictionary containing each cost for each edge

Functions:

- **def get_nr_of_vertices(self)**
  - returns an integer containing the number of vertices in the directed graph
- **def parse_keys(self)**
  - returns a copy of all vertex keys
- **def is_edge(self, vertex1, vertex2)**
  - returns true if there is an edge from vertex1 to vertex2, false otherwise
- **def get_in_degree(self, vertex)**
  - returns an integer representing the IN degree of the vertex x if the vertex is a valid one
- **def get_out_degree(self, vertex)**
  - returns an integer representing the OUT degree of the vertex if the vertex is a valid one
- **def parse_in_neighbours(self, vertex)**
  - returns a copy of all “in” neighbours of vertex if the vertex is a valid one
- **def parse_out_neighbours(self, x)**
  - returns a copy of all “out” neighbours of vertex if the vertex is a valid one
- **def add_edge(self, vertex1, vertex2, cost)**
  - adds an edge if the vertexes are valid and the edge doesn’t already exist
  - else it returns an exception with the specific message
- **def remove_edge(self, vertex1, vertex2)**
  - removes the edge (vertex1, vertex2) from the graph

  - if the edge doesn’t exist then it returns an exception with the specific message
- **def get_cost(self, vertex1, vertex2)**
  - returns the cost of the (vertex1, vertex2) edge
  - if the edge is not a valid one an exception is given with the specific message
- **def add_vertex(self, vertex)**
  - adds a vertex to the graph, as an isolated vertex
  - if the vertex already exists in the graph an exception is returned
- **def remove_vertex(self, vertex)**
  - removes a vertex from the graph
  - if the vertex doesn’t exist in the graph, then an exception is returned
- **def copy_the_graph(self)**
  - creates a copy of the graph and also returns it
- **def return_edges(self)**
  - returns an iterable containing all the edges
- **def return_costs(self)**
  - returns an iterable containing all the costs
- **def change_edge_cost(self, vertex1, vertex2, cost)**
  - changes the cost of an edge (vertex1, vertex2)
  - if the edge doesn’t exist in the graph, it returns an exception
- **def isolated_vertices(self)**
  - returns an iterable containing all the isolated vertices
- **def change_edge_cost(self, vertex1, vertex2, cost)**
  - modifies the cost of an edge (vertex1, vertex2)
  - if the edge doesn’t exist in the graph an exception is returned

We have another new class: *RandomGraph*

Fields:

- **vertices**- being the number of vertices for the random generated graph
- **edges**- being the number of edges for the random generated graph
- **randomG**- is an object of the previous class, Graph, with a given number of vertices
- **generate**- a function that generates randomly a directed graph with 2 parameters: vertices and edges

Functions:

- **__generate(self, vertices, edges)**:
  - a function that generates randomly a directed graph with 2 parameters: vertices and edges
  - we use the library random for generating them
- **print_graph(self):**
  - prints the graph's characteristics

We have a new class: *<u>Run</u>*

Fields:

- fileName- being the name of the file from which we read the directed graph
- commands- a dictionary, with keys from 1-15 having as values methods for the directed graph.

As functions used, we have the ones implemented in the class **Graph** in user interface mode.

We have 5 modules: **graph** (containing the class Graph), **randomGraph** (containing the class RandomGraph), **main**(containing the class Run), **exceptions** (containing the class graphException, used for returning exceptions), and **menu** (containing the function print_menu()) .