Bolog Riccardo - 911

DirectedGraph class documentation

Methods:

__init__(self, no_vertices: int)

- ➔ ***Precondition: no_vertices has to be an integer***
- ➔ ***This is the class constructor, it initializes a directed graph with the given number of vertices, all vertices are isolated in the first place***
- ➔ ***It has 3 dictionaries:***
  - ***dict_in: has as key a vertex and the value is a list of it's inbound neighbours***
  - ***dict_out: has as key a vertex and the value is a list of it's outbound neighbours***
  - ***dict_cost: has as key a tuple containing an edge and the value is the cost of that edge***
- ➔ ***It also has 2 more attributes:***
  - ***vertices: the current number of vertices***
  - ***edges: the current number of edges (initially 0)***

get_graph_from_file(self, file_name: str)

- ➔ ***Precondition: file_name has to be an existing file with the following format:***

numberOfVertices numberOfEdges
vertex1 vertex2 edgeCost1
.
.
.
vertexX vertexY edgecostN

- ➔ ***This method opens the file in reading mode and processes the lines from the file, adding every edge it finds***

get_graph_from_file_format_2(self, file_name: str)

➔ ***Precondition: file_name has to be an existing file with the following format:***

vertex1 vertex2 edgeCost
. . .
vertexN -1 (if it's an isolated vertex)
. . .

➔ ***This method opens the file in reading mode and processes the lines from the file, adding the vertices if they do not exist already in the graph, and adding the edges afterwards, if it's not an isolated vertex***

copy_graph(self)

➔ ***This method returns a deepcopy of the current graph***

get_number_of_vertices(self)

➔ ***This method returns the number of vertices of the current graph***

get_number_of_edges(self)

➔ ***This method returns the number of edges of the current graph***

parse_vertices(self)

➔ ***This method makes and returns the list of vertices obtained by extracting the keys of the dict_in attribute***

is_edge(self, edge: tuple)

➔ ***This method verifies if an edge exists in the current graph and returns true if it's the case, false otherwise***

is_vertex(self, vertex: int)

➔ ***This method verifies if a vertex exist in the current graph and returns true if it's the case, false otherwise***

get_in_degree(self, vertex: int)

➔ ***Precondition: vertex must be valid in the graph***
➔ ***Returns the in degree of the specified vertex***
➔ ***Raises an exception if the vertex does not exist***

get_out_degree(self, vertex: int)

➔ ***Precondition: vertex must be valid in the graph***
➔ ***Returns the out degree of the specified vertex***
➔ ***Raises an exception if the vertex does not exist***

parse_outbound_edges(self, vertex: int)

➔ ***Precondition: vertex must be valid in the graph***
➔ ***Returns the value of dict_out[vertex], the list of outbound edges***
➔ ***Raises an exception if the vertex does not exist***

parse_inbound_edges(self, vertex: int)

➔ ***Precondition: vertex must be valid in the graph***
➔ ***Returns the value of dict_in[vertex], the list of inbound edges***
➔ ***Raises an exception if the vertex does not exist***

get_cost(self, edge: tuple)

➔ ***Precondition: edge must be valid in the graph***
➔ ***Returns the cost of the specified edge***
➔ ***Raises an exception if the edge does not exist***

set_cost(self, edge: tuple, cost: int)

➔ ***Precondition: edge must be valid in the graph***
➔ ***This method sets the cost of the specified edge to the value of cost***

add_edge(self, edge: tuple, cost: int)

➔ ***Precondition: edge should not be in the graph and the vertices of the edge should be in the graph***
➔ ***This method adds a specified edge, increasing the number of edges also***
➔ ***Raises an exception if the edge is in the graph already or if one of the two vertices of the edge are not part of the graph***

remove_edge(self, edge: tuple)

➔ ***Precondition: edge should be in the graph***

➔ *This method removes the specified edge, decreasing the number of edges also*
➔ *Raises an exception if the edge is not part of the graph*

add_vertex(self, vertex: int)

➔ *Precondition: vertex should not be part of the graph*
➔ *This method adds a specified vertex, making it isolated, increasing the number of vertices also*
➔ *Raises an exception if the vertex is part of the graph already*

remove_vertex(self, vertex: int)

➔ *Precondition: vertex should be part of the graph*
➔ *This method removes the specified vertex and all his occurrences in the potential edges it might be part of, decreasing the number of vertices also*
➔ *Raises an exception if the vertex is not part of the graph*

dict_cost(self)

➔ *This is a getter property, returns the dict_cost*

menu()

➔ *This is a static method of the class, prints on the screen the menu with implemented functionalities*

External functions:

get_vertices_from_file(file_name: str)

➔ *Precondition: file_name is valid and has the first format*
➔ *Returns the number of vertices specified in the file*

get_edges_from_file(file_name: str)

➔ *Precondition: file_name is valid and has the first format*
➔ *Returns the number of edges specified in the file*

generate_random_graph(no_vertices: int, no_edges: int)

➔ *Precondition:* $no\_edges \leq no\_vertices^2$

➔ *This function creates and returns a randomly generated graph with the specified number of vertices and number of edges*

➔ *Raises an exception if the precondition is not met*

write_graph_to_file(graph: DirectedGraph, filename: str)

➔ *Precondition: graph is valid*

➔ *This function opens the file in write mode and it translates the specified graph into the file using the second format*

run()

➔ *This is the main function of the app, it starts everything, printing the menu and getting the user input, also catches the exceptions and prints them if it's the case, has a loop that ends when user presses the exit option*