Graphs Laboratory 1 Homework Documentation

Moldovan Gelu-Daniel

Mar 30, 2022

CONTENTS:

CHAPTER

ONE

SRC

1.1 main module

1.2 DirectedGraph module

class DirectedGraph.**DirectedGraph**(*vertices*)

Bases: object

add_edge(*x*, *y*, *c*)

This function adds an edge to the graph.

Parameters

- **x** (*integer*) – the first vertex
- **y** (*integer*) – the second vertex
- **c** (*integer*) – the cost of the edge

Raises

- ***EdgeError*** – if the edge already exists
- ***VertexError*** – if at least one vertex doesn't exist.

add_vertex(*vertex*)

This function adds a vertex to the graph.

Parameters **vertex** (*integer*) – vertex that is going to be added.

Raises ***VertexError*** – if the vertex already exists.

check_vertex(*vertex*)

This function checks if a vertex is in the list of vertices.

Parameters **vertex** (*integer*) – vertex that is going to be checked

Returns True if the vertex is found, false otherwise

create_random_graph(*vertices*, *edges*)

This function creates a random graph.

Parameters

- **vertices** (*integer*) – number of vertices for the new graph.
- **edges** (*integer*) – number of edges for the new graph.

delete_edge(*first_vertex*, *second_vertex*)

This function deletes an edge from the graph.

Parameters

- **first_vertex** (*integer*) – first vertex of the edge.
- **second_vertex** (*integer*) – second vertex of the edge.

Raises

- ***EdgeError*** – if the edge doesn't exist.
- ***VertexError*** – if at least one vertex doesn't exist.

delete_vertex(*vertex*)
This function deletes a vertex from the graph.

Parameters **vertex** (*integer*) – vertex that is going to be deleted.

Raises ***VertexError*** – if the vertex doesn't exist.

find_all_the_isolated_vertices()
This function returns a list with all the isolated vertices.

Returns list with isolated vertices.

find_edge_for_2_vertices(*first_vertex*, *second_vertex*)
This function returns the index for a given edge.

Parameters

- **first_vertex** (*integer*) – first vertex of edge.
- **second_vertex** (*integer*) – second vertex of edge

Returns the i-th element from the list, if the edge was found. -1 otherwise.

get_edges()
This function returns the list of edges with costs.

Returns list with the edges and the costs for each edge.

in_neighbours(*x*)
This function returns the in neighbours for a given vertex.

Parameters **x** (*integer*) – the vertex for which we want to know the in neighbours

Raises ***VertexError*** – if the vertex is not in the list of vertices.

Returns a copy of the list of in neighbours for the given vertex.

is_edge(*x*, *y*)
This function checks if there is an edge between 2 vertices.

Parameters

- **x** (*integer*) – the first vertex
- **y** (*integer*) – the second vertex

Raises ***VertexError*** – if the vertex is not in the list of vertices.

Returns True if there is an edge, False otherwise.

isolated_vertex(*vertex*)
This function checks if a vertex is isolated.

Parameters **vertex** (*integer*) – vertex for which we want to check if it is isolated.

Raises ***VertexError*** – if the vertex is not in the list of vertices.

Returns True if the vertex is an isolated one, False otherwise.

load_graph_from_a_text_file (*file_name*)
This function loads the graph from a given text file and it creates a graph with the informations from file.

Parameters **file_name** (*string*) – name of the file

Raises ***FileError*** – if the file doesn't exist.

out_neighbours (*x*)
This function returns the out neighbours for a given vertex.

Parameters **x** (*integer*) – the vertex for which we want to know the out neighbours

Raises ***VertexError*** – if the vertex is not in the list of vertices.

Returns a copy of the list of out neighbours for the given vertex.

save_graph_to_a_text_file (*file_name*)
This function save the graph in a text file. On the first line is going to be the number of vertices. On the second line is going to be the isolated vertices. And then on the next lines are going to be the edges with the cost.

Parameters **file_name** (*string*) – name of the file.

start (*vertices*)
This function creates the inbound and outbound list for each vertex.

Parameters **vertices** (*a list of ints.*) – a list of initial vertices.

update_cost (*first_vertex*, *second_vertex*, *new_cost*)
This function updates the cost for a given edge.

Parameters

- **first_vertex** (*integer*) – first vertex of the edge.
- **second_vertex** (*integer*) – second vertex of the edge
- **new_cost** (*integer*) – the new cost for the given edge

Raises

- ***EdgeError*** – if the edge doesn't exist
- ***VertexError*** – if at least one vertex doesn't exist.

vertices ()
This function returns the list of vertices.

Returns list of vertices.

1.3 Exception module

exception Exception.**EdgeError**
Bases: *Exception.Error*

exception Exception.**Error**
Bases: *Exception*

exception Exception.**FileError**
Bases: *Exception.Error*

exception Exception.**VertexError**
Bases: *Exception.Error*

exception Exception.**VertexOrEdge**
Bases: *Exception.Error*

exception Exception.**YesOrNo**
Bases: *Exception.Error*

1.4 Tests module

class Tests.**TestDirectedGraph**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp() → None
Hook method for setting up the test fixture before exercising it.

tearDown() → None
Hook method for deconstructing the test fixture after testing it.

test_add_edge()

test_add_vertex()

test_delete_edge()

test_delete_vertex()

test_find_all_the_isolated_vertices()

test_find_edge_for_2_vertices()

test_in_neighbours()

test_isolated_vertex()

test_out_neighbours()

test_update_cost_of_an_edge()

1.5 UI module

class UI.**UI**
Bases: object

add()
This function connects add_vertex and add_edge. :raises VertexOrEdge: if the input is not vertex or edge.

add_edge()
This function checks the input and it adds the edge to the graph.

add_vertex()
This function checks the input and it adds the vertex to the graph.

Returns

create_random_graph()
This function creates a random graph and also validates the input and it checks if it's possible to create a graph with the given number of vertices and edges.

delete()
This function connects delete_vertex and delete_edge :raises VertexOrEdge: if the input is not vertex or edge.

delete_edge()
This function checks the input and deletes the edge from the graph.

delete_vertex()
This function checks the input and deletes the vertex from the graph.

get_all_the_isolated_vertices()
This function prints the list of isolated vertices.

get_degree()
This function counts the in and out degree for a given vertex. Also it validates the input.

get_edges_and_cost()
This function prints the list of edges with the costs.

get_number_of_vertices()
This function prints the number of vertices and also the list of vertices if the user wants to. :raise YesOrNo: if the user doesn't answer with yes or no.

load_graph()
This function loads the graph from a given text file.

print_both()
This function prints the outbounds and the inbounds for the graph.

print_inbound()
This function prints the inbounds for the graph.

print_inbounds_for_a_vertex(*vertex*)
This function prints the inbounds of a vertex.

Parameters **vertex** (*integer*) – the vertex for which we want to see the inbounds.

print_outbounds()
This function prints the outbounds for the graph.

print_outbounds_for_a_vertex(*vertex*)
This function prints the outbounds of a vertex.

Parameters **vertex** (*integer*) – the vertex for which we want to see the outbounds.

print_specific()
This function connects the inbounds and outbounds print.

run_console()
This function is the heart of the UI. It gets the input and it connects all the functions from UI.

save_graph()
This function saves the graph to a file.

update_cost()
This function validates the input and it updates the cost for a given edge.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

T

U

V

Y