
Laboratory 5 graphs Documentation

Rapeanu George - Alexandru

May 21, 2022

CONTENTS:

1	python	1
1.1	Graph module	1
1.2	GraphTests module	4
1.3	UI module	4
2	Indices and tables	5
	Python Module Index	7
	Index	9

1.1 Graph module

class `Graph.Graph` (*vertices*, *edges*)

Bases: `object`

add_edge (*x*, *y*, *z*)

This function adds the edge from *x* to *y* to the graph

Parameters

- ***x*** (*str*) – the first vertex
- ***y*** (*str*) – the second vertex
- ***z*** (*int*) – the cost

Raises

- **Exception** – if types do not follow the specification
- **Exception** – if nodes do not exist
- **Exception** – if edge already exists

add_vertex (*x*)

This function adds the vertex *x* to the graph

Parameters ***x*** (*str*) – the vertex to be added

Raises

- **Exception** – if *x* is not string
- **Exception** – if *x* already exists

copy ()

This function retrieves a copy of the current graph

Returns a Graph copy

get_edge_cost (*x*, *y*)

This function returns the cost of the edge from *x* to *y*

Parameters

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

Returns the cost of the edge from *x* to *y*

Raises `Exception` – if there is no edge from `x` to `y`

`get_in_degree(x)`

This function returns the in degree of a vertex

Parameters `x (str)` – the vertex

Returns the in degree of the vertex `x`

Raises `Exception` – if `x` doesn't exist

`get_out_degree(x)`

This function returns the out degree of a vertex

Parameters `x (str)` – the vertex

Returns the out degree of the vertex `x`

Raises `Exception` – if `x` doesn't exist

`is_edge(x, y)`

This function returns True if the edge `x->y` exists, false otherwise

Parameters

- `x (str)` – the first vertex
- `y (str)` – the second vertex

Returns True if an edge exists, false otherwise

Raises `Exception` – if `x` or `y` are not vertices

`modify_edge_cost(x, y, z)`

This function modifies the cost of the edge from `x` to `y`

Parameters

- `x (str)` – the first vertex
- `y (str)` – the second vertex
- `z (int)` – the new cost

Raises `Exception` – if there is no edge from `x` to `y`

`parse_inbound_edges(x)`

This function returns an iterable of deepcopied vertices

Parameters `x` – the vertex for which to retrieve the iterator

Returns iterator to a deepcopied list of inbound vertices

Raises `Exception` – if the vertex doesn't exist

`parse_outbound_edges(x)`

This function returns an iterable of deepcopied vertices

Parameters `x` – the vertex for which to retrieve the iterator

Returns iterator to a deepcopied list of outbound vertices

Raises `Exception` – if the vertex doesn't exist

`parse_vertices()`

This function returns an iterable containing nodes

The nodes are deepcopied, in order to avoid being modified from the outside :return: iterator through a list of deepcopied nodes

remove_edge (*x*, *y*)

This function removes the edge from *x* to *y* from the graph

Parameters

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

Raises Exception – if edge already exists

remove_vertex (*x*)

This function removes the vertex *x* from the graph

Parameters **x** (*str*) – the vertex to be removed

Raises Exception – if *x* doesn't exist

Graph.get_hamiltonian_path (*graph*)

This function get a hamiltonian path in the graph

Parameters **graph** (*Graph*) – the graph

Returns tuple of (the cost, a list of nodes which represents the path)

Graph.random_graph (*n*, *m*)

This function creates a random graph with specified number of vertices and edges

Parameters

- **n** (*int*) – the number of vertices
- **m** (*int*) – the number of edges

Returns a graph with specified parameters

Raises Exception – if invalid parameters

Graph.read_graph (*filename*)

This function reads a graph from a file. It supports 2 formats .txt and .modified.txt

In case of .txt, the file is supposed to look like this:

On the first line, the number *n* of vertices and the number *m* of edges; On each of the following *m* lines, three numbers, *x*, *y* and *c*, describing an edge: the origin, the target and the cost of that edge.

In case of .modified.txt, the file is supposed to look like this:

On the first line, the number *n* of vertices and the number *m* of edges On the second line, a list of the *n* vertices separated by space On each of the following *m* lines, three numbers, *x*, *y* and *c*, describing an edge: the origin, the target and the cost of that edge.

Parameters **filename** (*str*) – the file from which to read(name, relative path or absolute path)

Returns *Graph*

Raises Exception – in case of invalid format

Graph.write_graph (*filename*, *graph*)

This function writes a graph from a file. It supports 1 format .modified.txt

On the first line, the number *n* of vertices and the number *m* of edges On the second line, a list of the *n* vertices separated by space On each of the following *m* lines, three numbers, *x*, *y* and *c*, describing an edge: the origin, the target and the cost of that edge.

Parameters

- **filename** (*str*) – the filename to which to read(name, relative path or absolute path), MUST end in .modified.txt
- **graph** (*Graph*) – the graph to be written

Raises **Exception** – if invalid data

1.2 GraphTests module

1.3 UI module

UI.display_edges (*edges*)

This function displays a given list of edges

Parameters **edges** (*list*) – list of edges represented as tuples

Returns None

UI.display_vertices (*vertices*)

This function displays the given vertices

Parameters **vertices** (*list*) – the vertices

Returns None

UI.main ()

The main of the program

Returns None

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

g

Graph, 1

u

UI, 4

INDEX

A

`add_edge()` (*Graph.Graph method*), 1
`add_vertex()` (*Graph.Graph method*), 1

C

`copy()` (*Graph.Graph method*), 1

D

`display_edges()` (*in module UI*), 4
`display_vertices()` (*in module UI*), 4

G

`get_edge_cost()` (*Graph.Graph method*), 1
`get_hamiltonian_path()` (*in module Graph*), 3
`get_in_degree()` (*Graph.Graph method*), 2
`get_out_degree()` (*Graph.Graph method*), 2
`Graph` (*class in Graph*), 1
`Graph` (*module*), 1

I

`is_edge()` (*Graph.Graph method*), 2

M

`main()` (*in module UI*), 4
`modify_edge_cost()` (*Graph.Graph method*), 2

P

`parse_inbound_edges()` (*Graph.Graph method*),
2
`parse_outbound_edges()` (*Graph.Graph method*),
2
`parse_vertices()` (*Graph.Graph method*), 2

R

`random_graph()` (*in module Graph*), 3
`read_graph()` (*in module Graph*), 3
`remove_edge()` (*Graph.Graph method*), 2
`remove_vertex()` (*Graph.Graph method*), 3

U

`UI` (*module*), 4

W

`write_graph()` (*in module Graph*), 3