
Lab1 Documentation

Gabor Ioana

Mar 26, 2023

CONTENTS

| | |
|---------------------|---|
| Python Module Index | 7 |
| Index | 9 |

class graph.Graph

Represents a directed graph.

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

Adds an edge to the graph. Preconditions: both of its vertices must exist, but the edge must not be present in the current graph.

:param cost: the cost of the added edge :param edge: tuple of 2 vertices, the added edge :return: nothing
:raises Exception: if one of its vertices doesn't exist, if the edge already exists or if the cost is not integer

add_vertex(*vertex: str*)

Adds a vertex to the graph. Preconditions: the vertex must not exist in the current graph.

Parameters

vertex – given vertex

Returns**Raises**

Exception – if the vertex already exists

classmethod build_graph_from_given_vertices_and_edges(*vertices, edges_with_costs*)

Builds a graph from a list of vertices and a list of edges with costs.

Parameters

- **vertices** – list of vertices
- **edges_with_costs** – list of tuples of 3 parameters - (first vertex, second vertex, cost)

Returns

created graph

Raises

Exception – if the list of received edges is invalid or if the vertices are not all strings

classmethod build_random_graph(*number_of_vertices: int, number_of_edges: int*)

Builds a random graph with a specified number of vertices and number of edges. Preconditions: the number of edges should be smaller than or equal to the square of the number of vertices.

Parameters

number_of_vertices – integer

:param number_of_edges: integer :return: created graph :raises Exception: if the number of edges is too large for a graph with distinct edges

classmethod create_copy(*graph*)

Creates a deepcopy of a graph

Parameters

graph –

Returns

created graph

get_all_edges()

Gets the list of all the edges in this graph.

Returns

list of tuple of 2 elements: a tuple for the edge (another tuple of 2 vertices) and an integer

get_all_vertices()

Gets the list of all the vertices in this graph.

Returns

list of vertices

get_cost_of_edge(*edge: tuple*)

Gets the cost of an edge.

Parameters

edge – tuple of vertices

Returns

integer - the cost of the given edge

Raises

Exception – if the edge does not exist

get_in_degree_of_vertex(*vertex: str*)

Gets the in-degree of a vertex. The in-degree is the number of vertices that directly reach this vertex.

Parameters

vertex – given vertex

Returns

integer - the in degree

Raises

Exception – if the vertex does not exist

get_inbound_neighbours(*vertex*)

Gets the inbound neighbours of a vertex.

Parameters

vertex – given vertex

Returns

list of outbound neighbours

Raises

Exception – if the vertex does not exist

get_isolated_vertices()

Gets the list of all the isolated vertices

Returns

list of vertices

get_number_of_vertices()

Gets the number of vertices in this graph.

Returns

integer

get_out_degree_of_vertex(*vertex*)

Gets the out-degree of a vertex. The out-degree is the number of vertices that are directly reachable from this vertex.

Parameters

vertex – given vertex

Returns

integer - the out degree

Raises

Exception – if the vertex does not exist

get_outbound_neighbours(*vertex*)

Gets the outbound neighbours of a vertex.

Parameters

vertex – given vertex

Returns

list of outbound neighbours

Raises

Exception – if the vertex does not exist

is_edge(*edge: tuple*)

Checks if an edge is part of this graph. Preconditions: the edge must be a tuple of 2 vertices.

Parameters

edge – tuple of 2 edges

Returns

boolean

Raises

Exception – if the edge is not a tuple of 2 vertices

is_vertex(*vertex: str*)

Checks if a vertex is part of this graph.

Parameters

vertex – given vertex

Returns

boolean

remove_edge(*edge: tuple*)

Removes an edge.

Parameters

edge – tuple of 2 vertices

Returns

nothing

Raises

Exception – if the edge doesn't exist

remove_vertex(*vertex: str*)

Removes a vertex.

Parameters

vertex – given vertex

Returns

nothing

Raises

Exception – if the vertex doesn't exist

set_cost_of_edge(*edge: tuple, cost: int*)

Sets the cost of an edge.

Parameters

- **edge** – tuple of vertices
- **cost** – integer

Returns

integer - the cost of the given edge

Raises

Exception – if the edge doesn't exist or if the cost is not an integer

class graph_utils.GraphUtils

Helper methods used for reading and writing graphs to files, in normal or modified format.

static read_graph_modified_format(*filename*)

Reads a graph in “modified format” from a given filename.

Preconditions: the file must exist, the filename must end with “modified.txt” and the graph should be in the valid format.

The “modified” format must obey the following rules: on the first line of the file, there are two numbers, separated by space: the number of vertices(*n*) and the number of edges(*m*). On the second line, there is the list of isolated vertices. On the following *m* lines, there are three numbers that describe each of the *m* edges: the starting vertex, the ending vertex and the cost of the edge.

Parameters

filename – string

Returns

Graph

Raises

Exception – if the graph is invalid, if the file doesn't exist, if the filename doesn't end with “modified.txt” or if other file-related errors occurred

static read_graph_normal_format(*filename*)

Reads a graph in “normal format” from a given filename.

Preconditions: the file must exist, the graph must be in the valid format.

The “normal” format must obey the following rules: on the first line of the file, there are 2 integers, separated by space: the number of vertices (*n*) and the number of edges (*m*). On the next *m* lines, there are three numbers that describe each of the *m* edges: the starting vertex, the ending vertex and the cost of the edge.

Parameters

filename – string

Returns

Graph

Raises

Exception – if the graph is invalid, if the file doesn't exist or if other file-related errors occurred.

static write_graph_modified_format(*filename: str, graph*)

Writes a graph in “modified format” to a given file.

Parameters

- **filename** – string
- **graph** – Graph

Raises

Exception – if the filename doesn’t end with “modified.txt” or if other file-related errors occurred.

static write_graph_normal_format(*filename, graph*)

Writes a graph in “normal format” to a given file.

Parameters

- **filename** – string
- **graph** – Graph

Raises

Exception – if output-related errors occurred

PYTHON MODULE INDEX

g

graph, 1

graph_utils, 4

INDEX

A

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

B

`build_graph_from_given_vertices_and_edges()`
 (*graph.Graph* class method), 1
`build_random_graph()` (*graph.Graph* class method), 1

C

`create_copy()` (*graph.Graph* class method), 1

G

`get_all_edges()` (*graph.Graph* method), 1
`get_all_vertices()` (*graph.Graph* method), 1
`get_cost_of_edge()` (*graph.Graph* method), 2
`get_in_degree_of_vertex()` (*graph.Graph* method),
 2
`get_inbound_neighbours()` (*graph.Graph* method), 2
`get_isolated_vertices()` (*graph.Graph* method), 2
`get_number_of_vertices()` (*graph.Graph* method), 2
`get_out_degree_of_vertex()` (*graph.Graph*
 method), 2
`get_outbound_neighbours()` (*graph.Graph* method),
 3
`graph`
 module, 1
`Graph` (class in *graph*), 1
`graph_utils`
 module, 4
`GraphUtils` (class in *graph_utils*), 4

I

`is_edge()` (*graph.Graph* method), 3
`is_vertex()` (*graph.Graph* method), 3

M

module
 graph, 1
 graph_utils, 4

R

`read_graph_modified_format()`
 (*graph_utils.GraphUtils* static method),
 4
`read_graph_normal_format()`
 (*graph_utils.GraphUtils* static method),
 4
`remove_edge()` (*graph.Graph* method), 3
`remove_vertex()` (*graph.Graph* method), 3

S

`set_cost_of_edge()` (*graph.Graph* method), 4

W

`write_graph_modified_format()`
 (*graph_utils.GraphUtils* static method),
 4
`write_graph_normal_format()`
 (*graph_utils.GraphUtils* static method),
 5