Laboratory 1 documentation

Grancea Alexandru Nicolae, group 913/2

By defining a class Graph, I aim to represent a directed graph with a number of edges and vertices. In order to properly do that, I am using two dictionaries that represent (I) the inbound neighbours of each vertex (II) the outbound neighbours of each vertex, and one tuple list that contains every edge.

The class Graph contains the following methods:

- __init__(self, n = 0, m = 0)

Constructor for class Graph, where n represents the numuber of vertices and m the number of edges, both of which are by default zero.

- def vertices(self)
  Returns the number of vertices in the graph.

- def edges(self)
  Returns the number of edges in the graph.

- def get_edge_cost(self, x, y)
  Returns the cost of the edge [x, y].

- def add_edge(self, x, y, z)

  Add the edge [x, y] with the cost z in the graph.If the edge already exists, or if the edge isn't valid, the function will throw an exception.

- def get_inner_edges(self, x)
  Returns the iterable inner edges for the vertex x.

- def get_outer_edges(self, x)
  Returns the iterable outer edges for the vertex x.

- def find_edge(self, x, y)

  Returns the edge if the edge [x, y] exists or None if the edge does not exist.

- def vertex_iterator(self)

  Returns a vertex iterator for the graph.

- def edge_iterator(self)

  Returns an edge iterator for the graph.

- def remove_edge(self, x, y)
  Removes the edge [x, y] if this edge exists by removing the inbound edge for y and the inbound edge for x. If the edge does not exist, the function will throw an exception.

- def remove_vertex(self, x)
  Removes the vertex x from the graph. First I delete all the inbound and outbound edges of the vertex x, after I delete it.If the vertex x does not exist, the function will throw an exception.

- def add_vertex(self)
  Add an empty vertex to the graph.

- def update_edge(self, x, y, z)
  Update the cost of the edge [x, y] with the new cost z. Throws an exception if the edge does not exist.

- def copy_graph(self)
  Returns a deep copy of the graph.

- def print_graph(self)
  Print the graph on the console.

- def get_inner_degree(self, x)
  Returns the inner degree of the vertex x.

- def get_outer_degree

Returns the outer degree of the vertex x.

Also, I have 3 more functions that generates a random graph, write a graph in a file and read a graph from a file:

- def generate_random_graph(n, m)
  Generates a random graph with n vertices and m edges.
- def read_graph(filename)
  Read a graph from a text file.
- def write_graph(graph, filename)
  Write a graph in a text file.

Implementation

I have two dictionaries that stores the inbound and outbound edges for every vertex that are initially empty:

```
self.__inner_edges = {}
self.__outer_edges = {}
for i in range(n):
    self.__inner_edges[i] = []
    self.__outer_edges[i] = []
```

,where n is the number of vertices.

To store the list of edges I use a Python list that stores a tuple with first vertex, second vertex and the cost.

```
self.__inner_edges[y].append(x)
self.__outer_edges[x].append(y)
self.__edges.append((x, y, z))
```