Ostapciuc Diana Miruna
Grupa 915, 31.03.2023

First laboratory homework

~implementation of a directed graph~

We shall define a class named “DirectedGraph” that holds in its “__init__” the following: two dictionaries (“graph” and “edge_ids”), the number of edges and the number of vertices. The graph will be implemented by making use of the “graph” dictionary, as its key holds the initial vertex of an edge. The lists attached to said key contains the second vertex of the edge, the weight, and the edge id.

```
class DirectedGraph:
    def __init__(self):
        self.graph = {}
        self.edge_ids = {}
        self.nr_of_edges = 0
        self.nr_of_vertices = 0
```

The class “DirectedGraph” will provide the following methods:

- adding a vertex/ an edge (adds the vertex as key, the edge with its contents to a list of said key)

```
def adding_a_vertex(self, v):
def adding_an_edge(self, v1, v2, weight):
```

- removing a vertex/ an edge

```
def removing_a_vertex(self, v):
def removing_an_edge(self, v1, v2):
```

- __str__ (configured for printing)
- getters: for number of vertices, the list of vertices, the edge’s endpoints and the weight of specific edges and the degree of specific vertices (and returns them)

```
def get_number_of_vertices(self):
def get_list_of_vertex(self):
def out_degree(self, v):
def in_degree(self, v):
def get_edge_endpoints(self, edge_id):
def get_edge_weight(self, v1, v2):

```

- setters: it modifies the cost of a specific edge

```
def set_edge_weight(self, v1, v2, weight):
```

- parsers for inbound/outbound vertices (returns a list of the inbound/outbound vertices of the given one)

```
def parse_outbound_vertex(self, vertex):
def parse_inbound_vertex(self, v):
```

The “edge_id” dictionary holds as key the edges we have in our graph, and the value that it holds is the id in itself. Through the menu that has been set up, we also have the ability of reading and writing graphs from/to text files and creating a randomized graph, along with making a copy of the graph. To create the copy, we use the “deepcopy” function.

To create a random graph, we first receive the number of vertices and edges as parameters. After that, we check to see if the creation of such graph is possible, and, if it is, we start randomizing our possibilities. We randomize the vertices we need, checking as to not add the same once twice. After this condition is met, we add them to the “graph” dictionary as keys, and start generating the second vertex. The cost is a randomized number from 1-10.

All of these commands are put together in a user interface, that is accessed throughout a menu.