Practical Work No. 1

This code is a Python script that implements a Graph data structure and provides methods for reading and writing a graph to a file, generating a random graph, and manipulating the graph.

We shall define 2 classes: “UI”- for the user interface ; “Graph”- for the graph operations. In addition we need 2 exception classes called: EdgeError and VertexError.

The `read_file` function takes a file path as input and reads a graph from the file. The file should contain the number of vertices, the number of edges, and each edge's start vertex, end vertex, and weight. The function returns a Graph object.

The `write_file` function takes a file path and a Graph object as inputs and writes the graph to the file. The file format is the same as the input file format for `read_file`.

The `random_graph` function takes the number of vertices and the number of edges as inputs and generates a random graph with the specified number of vertices and edges. The function returns a Graph object.

The Graph class provides methods for working with a graph. The class has the following attributes:

- `_vertices`: a set of vertices in the graph
- `_neighbours`: a dictionary containing each vertex's outbound neighbours
- `_transpose`: a dictionary containing each vertex's inbound neighbours

- `_cost`: a dictionary containing the cost of each edge

The class provides the following methods for working with the graph:

- `vertices_iterator(self)`: This method returns an iterator over the set of all vertices in the graph.
- `neighbours_iterator(self, vertex)`: This method returns an iterator over the set of all outbound neighbours of a given vertex. The parameter vertex is the vertex for which the neighbours are being sought.
- `transpose_iterator(self, vertex)`: This method returns an iterator over the set of all inbound neighbours of a given vertex. The parameter vertex is the vertex for which the inbound neighbours are being sought.
- `edges_iterator(self)`: This method returns an iterator over the set of all edges in the graph. Each edge is represented as a tuple (vertex1, vertex2, cost) where vertex1 and vertex2 are the two endpoints of the edge and cost is the cost of the edge.
- `is_vertex(self, vertex)`: This method returns True if the given vertex is in the graph and False otherwise.

- `is_edge(self, vertex1, vertex2)`: This method returns True if there exists an edge between the two vertices vertex1 and vertex2 in the graph and False otherwise.

- `count_vertices(self)`: This method returns the number of vertices in the graph.
- `count_edges(self)`: This method returns the number of edges in the graph.

- `in_degree(self, vertex)`: This method returns the number of edges in the graph that have the given vertex as their endpoint.

- `out_degree(self, vertex)`: This method returns the number of edges in the graph that have the given vertex as their starting point.

- `get_edge_cost(self, vertex1, vertex2)`: This method returns the cost of the edge between the two vertices vertex1 and vertex2 in the graph.

- `set_edge_cost(self, vertex1, vertex2, new_cost)`: This method sets the cost of the edge between the two vertices vertex1 and vertex2 in the graph to the given value new_cost.

- `add_vertex(self, vertex)`: This method adds a new vertex to the graph. If the vertex already exists in the graph, an exception is raised.

- `add_edge(self, vertex1, vertex2, cost)`: This method adds a new edge between the vertices vertex1 and vertex2 in the graph with the given cost. If either of the vertices does not exist in the graph, an exception is raised. If the edge already exists in the graph, its cost is updated to the new value.

- `remove_vertex(self, vertex)`: This method removes a vertex from the graph along with all its incident edges.

- `remove_edge(self, vertex1, vertex2)`: This method removes the edge between the vertices vertex1

The Graph class uses an adjacency list representation to store the graph. The add_edge method adds an edge to the graph by updating the _neighbours, _transpose, and _cost dictionaries. The add_vertex method adds a vertex to the _vertices set.

If an invalid vertex or edge is used as an argument in a method, an exception is raised with a corresponding error message.

Overall, this code provides a useful tool for working with graphs in Python, including generating and reading graphs from files, manipulating graphs, and writing graphs to files.

Birou Rares