Directed Graph Documentation

Dumitrascu Constantin-Alexadru
Group 913/2

1 Initialization Method

The initialization method `__init__()` is responsible for creating a directed graph object. It takes an optional parameter `vertices`, which specifies the number of vertices in the graph. If `vertices` is not provided, it defaults to 0.

1.1 Parameters

- `vertices`: An integer representing the number of vertices in the graph. Default value is 0.

1.2 Attributes

- `_vertices`: A set containing the vertices of the graph.
- `_g`: A dictionary representing the adjacency list of each vertex.
- `_g_inverse`: A dictionary representing the inverse adjacency list of each vertex.
- `_cost`: A dictionary storing the costs of edges.

1.3 Initialization Procedure

The initialization method initializes the graph object with the following steps:

1. Create an empty set `_vertices` to store vertices.
2. Create empty dictionaries `_g` and `_g_inverse` to store adjacency lists and inverse adjacency lists, respectively.
3. Create an empty dictionary `_cost` to store edge costs.
4. If the `vertices` parameter is provided, iterate over the range of `vertices`:
   (a) Add each vertex to the `_vertices` set.
   (b) Initialize empty sets for the adjacency list and inverse adjacency list of each vertex in `_g` and `_g_inverse`, respectively.

2 Graph Class Methods

1. **vertices_iterator():**
   - Iterates over vertices in the graph.
2. **out_neighbours_iterator(vertex:int):**
   - Iterates over outgoing neighbors of a given vertex.
3. **in_neighbours_iterator(vertex:int):**
   - Iterates over incoming neighbors of a given vertex.
4. **edges_iterator():**
   - Iterates over edges in the graph along with their costs.
5. **is_vertex(vertex):**
   - Checks if a vertex exists in the graph.
6. **is_edge(vertex1, vertex2):**
   - Checks if an edge exists between two vertices.
7. **vertices_num():**
   - Returns the total number of vertices in the graph.
8. **edges_num():**
   - Returns the total number of edges in the graph.
9. **out_degree(vertex):**
   - Returns the out-degree of a vertex.
10. **in_degree(vertex):**
    - Returns the in-degree of a vertex.
11. **get_edge_cost(vertex1, vertex2):**
    - Returns the cost of the edge between two vertices.
12. **set_edge_cost(vertex1, vertex2, new_cost):**
    - Sets a new cost for the specified edge.
13. **add_edge(vertex1, vertex2, edge_cost):**
    - Adds a new edge between two vertices with the given cost.
14. **remove_edge(vertex1, vertex2):**

- Removes the specified edge from the graph.

15. **add_vertex(vertex):**
    - Adds a new vertex to the graph.
16. **remove_vertex(vertex):**
    - Removes the specified vertex from the graph along with all incident edges.
17. **__deepcopy__():**
    - Returns a deep copy of the graph object.

3 File Operations

3.1 `write_file(graph, file_path)`

This function writes the contents of a graph object to a file.

3.1.1 Parameters

- `graph`: A graph object to be written to the file.
- `file_path`: A string representing the file path where the graph will be written.

3.1.2 Procedure

The function performs the following steps:

1. Opens the specified file in write mode.
2. Writes the number of vertices and edges of the graph to the file.
3. Iterates over each vertex in the graph:
    (a) Iterates over each outgoing neighbor of the current vertex.
    (b) Writes the edge information (source vertex, target vertex, and edge cost) to the file.

3.2 `read_file(file_path)`

This function reads a graph from a file and returns the corresponding graph object.

3.2.1 Parameters

- `file_path`: A string representing the file path from which the graph will be read.

3.2.2 Returns

A graph object representing the graph read from the file.

3.2.3 Procedure

The function performs the following steps:

1. Opens the specified file in read mode.
2. Reads the number of vertices and edges of the graph from the first line of the file.
3. Initializes a graph object with the number of vertices read.
4. Reads each edge information (source vertex, target vertex, and edge cost) from the file and adds it to the graph.
5. Returns the graph object.

4 Random Graph Generation

4.1 `random_graph(vertices_number, edges_number)`

This function generates a random graph with the specified number of vertices and edges.

4.1.1 Parameters

- `vertices_number`: An integer representing the number of vertices in the random graph.
- `edges_number`: An integer representing the number of edges in the random graph.

4.1.2 Returns

A graph object representing the randomly generated graph.

4.1.3 Procedure

The function performs the following steps:

1. Initializes an empty graph object.
2. Adds the specified number of vertices to the graph.
3. Generates the specified number of edges randomly:
   (a) Randomly selects two vertices.

(b) Checks if an edge already exists between the selected vertices.

(c) If no edge exists, adds a new edge with a random cost.

4. Returns the randomly generated graph.