Documentation Practical Work no. 1

`Graph()`

Class methods:

Represents a directed graph.

- **Initializer:**
  - **Args:**
    - `no_vertices=0`: number of vertices (default: 0)
    - `no_edges=0`: number of edges (default: 0)
  - **Behavior:**
    - Initializes the graph with the given number of vertices and edges. Each vertex is assigned an empty list for both inbound and outbound vertices.

Properties:

`vertices`

- **Type:** list
- **Description:** Contains the list of vertices in the graph.

`edges`

- **Type:** dict
- **Description:** Stores the edges of the graph as key-value pairs, where the key is a tuple `(i, j)` representing an edge from vertex `i` to vertex `j`, and the value is the cost of that edge.

`din`

- **Type:** dict
- **Description:** Represents the inbound vertices for each vertex in the graph.

`dout`

- **Type:** dict
- **Description:** Represents the outbound vertices for each vertex in the graph.

`numberOfEdges`

- **Type:** int
- **Description:** Represents the total number of edges in the graph.

`__str__()`

Returns a string representation of the graph, including its outbounds, inbounds, and edges.

- **Returns:** str
- **Description:** Formats and returns a string containing outbounds, inbounds, and edges of the graph. If no edges exist, it indicates so.

Service methods:

`read_file()`

Reads a graph from a file and stores it in the repository.

- **Args:**
  - `file_name`: the name of the file

`write_file()`

Writes the whole graph to the file, overwriting the previous content.

- **Args:**
  - `file_name`: the name of the file

`write_given_graph_to_file(graph: Graph, file_name: str)`

Writes a randomly generated graph to a file.

- **Args:**
  - `graph`: the graph to be written
  - `file_name`: the name of the file

`add_vertex(i)`

Adds a vertex to the graph if it does not already exist.

- **Args:**
  - `i`: "name" of the vertex

`remove_vertex(i)`

Removes a vertex from the graph if it exists.

- **Args:**
  - `i`: "name" of the vertex

`add_edge(i, j, cost)`

Adds an edge to the directed graph from i to j.

- **Args:**
  - `i`: first vertex (out)
  - `j`: second vertex (in)
  - `cost`: the cost of the edge

`remove_edge(i, j)`

Removes an edge from the graph.

- **Args:**
  - `i`: first vertex (out)
  - `j`: second vertex (in)

`is_vertex(i) -> bool`

Checks if a vertex exists in the graph.

- **Args:**
  - `i`: "name" of the vertex

`is_edge(i, j) -> bool`

Checks if an edge exists in the graph.

- **Args:**
  - `i`: first vertex (out)
  - `j`: second vertex (in)

`get_isolated_vertices() -> list`

Returns a list of isolated vertices.

`copy_graph() -> Graph`

Returns a copy of the current graph.

`generate_random_graph(no_vertices: int, no_edges: int) -> Graph`

Generates a random graph with a given number of vertices and edges.

- **Args:**
  - `no_vertices`: number of vertices
  - `no_edges`: number of edges

`get_vertices() -> list`

Returns a list of vertices.

`update_edge_cost(i: int, j: int, cost: int)`

Updates the cost of an edge.

- **Args:**
  - `i`: first vertex (out)
  - `j`: second vertex (in)
  - `cost`: the new cost

`in_degree_of_vertex(i: int) -> int`

Returns the in-degree of a vertex.

- **Args:**
  - `i`: "name" of the vertex

`out_degree_of_vertex(i: int) -> int`

Returns the out-degree of a vertex.

- **Args:**
  - `i`: "name" of the vertex

`number_of_vertices() -> int`

Returns the number of vertices in the graph.

`number_of_edges() -> int`

Returns the number of edges in the graph.

`get_inbounds_of_vertex(i: int) -> list`

Returns a list of inbounds of a vertex.

- **Args:**
  - `i`: "name" of the vertex

`get_outbounds_of_vertex(i: int) -> list`

Returns a list of outbounds of a vertex.

- **Args:**
  - `i`: "name" of the vertex

`Graph`

Represents a directed graph.

- **Initializer:**
  - **Args:**

    - `no_vertices=0`: number of vertices (default: 0)
    - `no_edges=0`: number of edges (default: 0)
  - **Behavior:**
    - Initializes the graph with the given number of vertices and edges. Each vertex is assigned an empty list for both inbound and outbound vertices.

`GraphService`

Provides functionalities for managing a graph.

- **Initializer:**
  - **Args:**
    - `repository: Repository`: an instance of the repository class
    - `file_name: str`: the name of the file to read/write
  - **Behavior:**
    - Initializes the service with the provided repository and file name.

Properties:

`repo`

- **Type:** Repository
- **Description:** The repository instance used by the service.

`fileName`

- **Type:** str
- **Description:** The name of the file used for reading/writing.

Methods:

`read_file()`

Reads a graph from a file and stores it in the repository.

- **Args:**
  - `file_name`: the name of the file

`write_file()`

Writes the whole graph to the file, overwriting the previous content.

- **Args:**
  - `file_name`: the name of the file

`write_given_graph_to_file(graph: Graph, file_name: str)`

Writes a randomly generated graph to a file.

- **Args:**
  - `graph`: the graph to be written
  - `file_name`: the name of the file

`add_vertex(i)`

Adds a vertex to the graph if it does not already exist.

- **Args:**
  - `i`: "name" of the vertex
- **Preconditions:**
  - The vertex must not already exist in the graph.

`remove_vertex(i)`

Removes a vertex from the graph if it exists.

- **Args:**
  - `i`: "name" of the vertex
- **Preconditions:**
  - The vertex must exist in the graph.

`add_edge(i, j, cost)`

Adds an edge to the directed graph from i to j.

- **Args:**
  - `i`: first vertex (out)
  - `j`: second vertex (in)
  - `cost`: the cost of the edge
- **Preconditions:**
  - The vertices must exist in the graph.

`remove_edge(i, j)`

Removes an edge from the graph.

- **Args:**
  - `i`: first vertex (out)
  - `j`: second vertex (in)
- **Preconditions:**

    - The edge must exist in the graph.

`is_vertex(i) -> bool`

Checks if a vertex exists in the graph.

- **Args:**
    - `i`: "name" of the vertex
- **Preconditions:**
    - The vertex must exist in the graph.

`is_edge(i, j) -> bool`

Checks if an edge exists in the graph.

- **Args:**
    - `i`: first vertex (out)
    - `j`: second vertex (in)
- **Preconditions:**
    - The edge must exist in the graph.

`get_isolated_vertices() -> list`

Returns a list of isolated vertices.

`copy_graph() -> Graph`

Returns a copy of the current graph.

`generate_random_graph(no_vertices: int, no_edges: int) -> Graph`

Generates a random graph with a given number of vertices and edges.

- **Args:**
    - `no_vertices`: number of vertices
    - `no_edges`: number of edges
- **Preconditions:**
    - The number of edges must be less than or equal to the maximum number of edges possible.

`get_vertices() -> list`

Returns a list of vertices.

`update_edge_cost(i: int, j: int, cost: int)`

Updates the cost of an edge.

- **Args:**
    - `i`: first vertex (out)
    - `j`: second vertex (in)
    - `cost`: the new cost
- **Preconditions:**
    - The edge must exist in the graph.

`in_degree_of_vertex(i: int) -> int`

Returns the in-degree of a vertex.

- **Args:**
    - `i`: "name" of the vertex
- **Preconditions:**
    - The vertex must exist in the graph.

`out_degree_of_vertex(i: int) -> int`

Returns the out-degree of a vertex.

- **Args:**
    - `i`: "name" of the vertex
- **Preconditions:**
    - The vertex must exist in the graph.

`number_of_vertices() -> int`

Returns the number of vertices in the graph.

`number_of_edges() -> int`

Returns the number of edges in the graph.

`get_inbounds_of_vertex(i: int) -> list`

Returns a list of inbounds of a vertex.

- **Args:**
    - `i`: "name" of the vertex
- **Preconditions:**
    - The vertex must exist in the graph.

`get_outbounds_of_vertex(i: int) -> list`

Returns a list of outbounds of a vertex.

- **Args:**
  - `i`: "name" of the vertex
- **Preconditions:**
  - The vertex must exist in the graph.