Documentation for practical work no. 1

Toarcaș Denisa-Alina

This is a Python code for implementing a directed graph data structure using three dictionaries. The three dictionaries are used to store information about the edges of the graph. Specifically, the **__dictionary_in** dictionary stores the inbound edges for each vertex, the **__dictionary_out** dictionary stores the outbound edges for each vertex, and the **__dictionary_cost** dictionary stores the cost of each edge.

The **TripleDictGraph** class defines several methods to manipulate the graph data structure. The **__init__** method initializes the three dictionaries to empty dictionaries.

The **number_of_vertices** method returns the number of vertices in the graph, which is the length of the **__dictionary_in** dictionary.

The **number_of_edges** method returns the number of edges in the graph, which is the length of the **__dictionary_cost** dictionary.

The **parse_vertices** method returns an iterable containing all the vertices of the graph. It uses the **.keys()** method on dictionaries to extract the keys (vertices) and returns them one by one using a generator.

The **parse_inbound_edges** method returns an iterable containing all the inbound edges of a given vertex. It retrieves the inbound edges from the **__dictionary_in** dictionary for the given vertex, and returns them one by one using a generator.

The **parse_outbound_edges** method returns an iterable containing all the outbound edges of a given vertex. It retrieves the outbound edges from the **__dictionary_out** dictionary for the given vertex, and returns them one by one using a generator.

The **parse_cost** method returns an iterable containing all the costs of the edges in the graph. It retrieves the keys (edges) from the **__dictionary_cost** dictionary, and returns them one by one using a generator.

The **in_degree** method returns the in-degree of a given vertex, which is the number of inbound edges for that vertex. It raises an exception if the vertex has no in-degree, which happens if the vertex is not a key in the **__dictionary_in** dictionary.

The **out_degree** method returns the out-degree of a given vertex, which is the number of outbound edges for that vertex. It raises an exception if the vertex has no out-degree, which happens if the vertex is not a key in the **__dictionary_out** dictionary.

The **is_vertex** method checks whether a given vertex is a vertex in the graph, which means it is a key in either the **__dictionary_in** or the **__dictionary_out** dictionary. It returns **True** if the vertex is a vertex in the graph, and **False** otherwise.

The **add_vertex** method adds a new vertex to the graph. It returns **True** if the vertex was added successfully, and **False** otherwise. It first checks whether the vertex is already a vertex in the graph, and adds it only if it is not.

The **remove_vertex** method removes a given vertex from the graph. It returns **True** if the vertex was removed successfully, and **False** otherwise. It first checks whether the vertex is a vertex in the graph, and removes it only if it is. It also removes all the inbound and outbound edges for the vertex from the **__dictionary_in** and **__dictionary_out** dictionaries, respectively, and removes all the costs for the edges involving the vertex from the **__dictionary_cost** dictionary.

The function **is_edge(x: int, y: int)** is a method of the **TripleDictGraph** class that checks if there is an edge between two vertices **x** and **y**. It returns **True** if there is an edge from **x** to **y**, and **False** otherwise. To determine if there is an edge between **x** and **y**, the method checks if **y** is in the inbound dictionary of vertex **x** and if **x** is in the outbound dictionary of vertex **y**. If both conditions are satisfied, then there is an edge from **x** to **y**, and **True** is returned; otherwise, **False** is returned. This method is used to ensure that an edge does not already exist before adding it to the graph or to check if an edge can be removed from the graph.

The function **add_edge(self, x: int, y: int, cost: int) -> bool**: Adds an edge from vertex **x** to vertex **y** with the specified **cost**. Returns **True** if the edge was added successfully, and **False** if the edge already exists in the graph.

The function **remove_edge(self, x: int, y: int) -> bool**: Removes the edge from vertex **x** to vertex **y** from the graph. Returns **True** if the edge was removed successfully, and **False** if the edge does not exist in the graph.

The function **modify_cost_of_edge(self, x: int, y: int, new_cost: int) -> bool**: Modifies the cost of the edge from vertex **x** to vertex **y** to the specified **new_cost**.

Returns **True** if the edge cost was modified successfully, and **False** if the edge does not exist in the graph.

The function **retrieve_edge(self, x: int, y: int) -> Optional[int]**: Returns the cost of the edge from vertex **x** to vertex **y** if it exists in the graph, and **None** otherwise.

The function **make_copy_of_graph(self) -> TripleDictGraph**: Returns a deep copy of the current graph.

The function **read_graph_from_file(filename: str) -> TripleDictGraph**: Reads a graph from a file with the given **filename**, in the following format:

- The first line contains two space-separated integers **n** and **m**, where **n** is the number of vertices in the graph and **m** is the number of edges in the graph.
- Each subsequent line contains three space-separated integers **u**, **v**, and **w**, representing an edge from vertex **u** to vertex **v** with weight **w**.
- If a line contains only one integer **u**, it represents an isolated vertex. Returns the graph as a **TripleDictGraph** object.

The function **write_graph_to_file(graph, filename: str) -> None**: Writes the graph to a file with the given **filename**, in the following format:

- The first line contains two space-separated integers **n** and **m**, where **n** is the number of vertices in the graph and **m** is the number of edges in the graph.
- Each subsequent line contains three space-separated integers **u**, **v**, and **w**, representing an edge from vertex **u** to vertex **v** with weight **w**.
- If a vertex is isolated, it is represented on a line by itself with only its vertex number. Raises an exception if the graph is empty (i.e., has no edges or vertices).