Directed Cost Graph

The DirectedCostGraph class is a Python implementation of a directed graph that supports edge costs. The class is initialized with three dictionaries, dict_in, dict_out, and dict_costs, that hold the graph's data.

Class methods:

- init(self) Creates a DirectedCostGraph object with empty dictionaries dict_in, dict_out, and dict_costs.
- is_vertex(self, vertex) Checks whether the specified vertex is in the graph. Returns True if it is, False otherwise.
- is_edge(self, edge_start, edge_end) Checks whether there is an edge from vertex "edge_start" to vertex "edge_end". Returns True if there is, False otherwise.
- add_vertex(self, new_vertex) Adds a new vertex to the graph by adding new key-value pairs to the in and out dictionaries. Returns True if successful, False if the vertex already exists in the graph.
- add_edge(self, edge_start, edge_end, cost: int = 0) Adds a new edge to the graph. Updates the in and out dictionaries of the vertices of the edge and adds a new edge to the cost dictionary. Returns True if the edge is successfully added, False if the edge already exists in the graph or if the vertices are not in the graph. Raises a GraphError if the vertices are not in the graph.
- remove_vertex(self, vertex) Removes the specified vertex from the graph by removing all related edges and deleting the vertex's key-value pairs from the in and out dictionaries. Returns True if the vertex is successfully removed, False otherwise.
- remove_edge(self, edge_start, edge_end) Removes the specified edge from the graph by removing it from the in and out dictionaries and the cost dictionary. Returns True if the edge is successfully removed, False otherwise.
- get_number_of_vertices(self) Returns the number of vertices in the graph.
- get_number_of_edges(self) Returns the number of edges in the graph.
- parse_all_vertices(self) Returns a list of all the vertices in the graph.

- parse_all_edges(self) Returns a list of all the edges in the graph as tuples of the form (edge_start, edge_end).
- parse_out_edges(self, vertex) Returns a list of all edges going out of the specified vertex as tuples of the form (vertex, successor). Raises a GraphError if the vertex is not in the graph.
- parse_in_edges(self, vertex) Returns a list of predecessors of the given vertex, intended to be iterated upon. Raises a GraphError if the vertex is not found in the graph.
- get_in_degree(self, vertex) Returns the number of predecessors of the given vertex. Raises a GraphError if the vertex is not found in the graph.
- get_out_degree(self, vertex) Returns the number of successors of the given vertex. Raises a GraphError if the vertex is not found in the graph.
- get_edge_cost(self, edge_start, edge_end) Returns the cost of the edge from edge_start to edge_end. Raises a GraphError if the edge is not found in the graph.
- modify_edge_cost(self, edge_start, edge_end, new_cost: int = 0) Modifies the cost of the edge indicated by the parameters. Raises a GraphError if there is no such edge.
- get_copy(self) Returns a deep copy of the DirectedCostGraph object.
- initialize_vertices(self, number_of_initial_vertices) Initializes a specified number of vertices in the DirectedCostGraph object. Each vertex is added to both dict_in and dict_out dictionaries.

Graph Controller

The read_graph_data function reads input data from a file with a given path and constructs a directed weighted graph according to the information in the file.

The file format has two possible conventions:

- Convention 1: The first line of the file contains the number of vertices and edges in the graph. The following lines contain information about each edge in the graph, with each line specifying the start vertex, end vertex, and edge cost.
- Convention 2: The first line of the file contains information about the first edge in the graph. The following lines contain either isolated vertices or information about edges in the graph.

The function first initializes an empty directed weighted graph. It then opens the input file at the given path and reads the first line to determine which file format is being used. If convention 1 is being used, the function reads the number of vertices and edges from the first line, initializes the graph with the appropriate number of vertices, and then reads each subsequent line to add the corresponding edges to the graph. If convention 2 is being used, the function reads the first line to add the first edge to the graph, and then reads each subsequent line to add isolated vertices or edges to the graph.

- :param file_path: The path of the input file to be read. :return: None. Changes the internal dict_in, dict_out, and dict_costs with values from the input file.
- Raises: - FileNotFoundError: if the input file is not found.

The write_graph_data function writes the graph to a file specified by the file_path parameter. The output file has a certain format which is documented in this function. If a vertex is an isolated node, it is written on a separate line with no edge information.

Parameters:

- file_path (str): the path of the file to be written. Default value is "data/input.txt".

File Format:

- If the graph contains only isolated nodes, the output file will have each vertex on a separate line.
- If the graph contains edges, the output file will start with a line that contains the number of vertices and edges in the graph. Following that, each line contains information about an edge, with the format "edge_start edge_end edge_cost".
- If an isolated node exists in the graph, it will be represented as an edge with the format "vertex --------> isolated node".

randomize_graph(self, number_of_vertices, number_of_edges, cost_range: tuple = (0, 99)) -> bool:

This function replaces the current graph with a random graph that has a specified number of vertices and edges. The cost of each edge in the graph will be a randomly generated integer within a specified range.

- :param number_of_vertices: the number of vertices that the random graph will have :param number_of_edges: the number of edges the random graph will have :param cost_range: a tuple containing the minimum and maximum values that the edge costs may take (default is (0, 99)) :return: False if the precondition that the number of edges is less than the number of vertices squared is not met, True if the random graph is successfully built and replaces the current graph.

make_graph_copy(self):

This function creates a copy of the current graph and stores it in self.__last_copy. If the copy is different from the current graph, the function returns True, otherwise it returns False.

- :return: True if a copy of the current graph is created and stored, False if the copy is identical to the current graph.

revert_to_last_copy(self):

This function replaces the current graph with the copy stored in self.__last_copy, if it exists and is different from the current graph. If the copy is the same as the current graph, the function returns False, otherwise it returns True.

- :return: True if the current graph is replaced by the stored copy, False if no copy exists or if the stored copy is identical to the current graph.