DirectedGraph class

`__init__(self, number_of_vertices=0)`

This is the constructor of the DirectedGraph class. It creates an empty directed graph with the given number of vertices. It initializes the internal dictionaries `__dict_in`, `__dict_out`, and `__costs`.

`dict_in`

This is a property that returns the dictionary representing the inbound edges of the graph.

`dict_out`

This is a property that returns the dictionary representing the outbound edges of the graph.

`costs`

This is a property that returns the dictionary representing the costs of the edges in the graph.

`is_vertex(self, vertex_to_check: int) -> bool`

This function checks if the given vertex is in the graph. It returns `True` if the vertex is in the graph, and `False` otherwise.

`add_vertex(self, vertex_to_add: int) -> None`

This function adds a vertex to the graph. It raises a `ValueError` if the vertex is already in the graph.

`remove_vertex(self, vertex_to_remove: int) -> None`

This function removes a vertex from the graph. It raises a `ValueError` if the vertex is not in the graph.

`is_edge(self, edge_to_check: tuple) -> bool`

This function checks if the given edge is in the graph. It returns `True` if the edge is in the graph, and `False` otherwise.

`add_edge(self, edge_to_add: tuple, cost: int) -> None`

This function adds an edge to the graph with the given cost. It raises a `ValueError` if the edge is already in the graph or the vertices of the edge are not in the graph.

`remove_edge(self, edge_to_remove: tuple) -> None`

This function removes an edge from the graph. It raises a `ValueError` if the edge is not in the graph.

`get_number_of_vertices(self) -> int`

This function returns the number of vertices in the graph.

`get_set_of_vertices(self) -> List[int]`

This function returns a list of all vertices in the graph.

`get_in_degree(self, vertex: int) -> int`

This function returns the in-degree of the given vertex. It raises a `ValueError` if the vertex is not in the graph.

`get_out_degree(self, vertex: int) -> int`

This function returns the out-degree of the given vertex. It raises a `ValueError` if the vertex is not in the graph.

`get_in_bound_edges(self, vertex: int) -> List[tuple]`

This function returns a list of all edges that have the given vertex as the endpoint of their inbound edge. It raises a `ValueError` if the vertex is not in the graph.

`get_out_bound_edges(self, vertex: int) -> List[tuple]`

This function returns a list of all edges that have the given vertex as the starting point of their outbound edge. It raises a `ValueError` if the vertex is not in the graph.

`get_all_edges(self) -> List[tuple]`

This function returns a list of all edges in the graph, along with their costs.

`make_copy(self) -> DirectedGraph`

This function returns a copy of the graph.

`read_graph_from_file_1(filename: str) -> DirectedGraph`

Reads a directed graph from a text file and returns it as a `DirectedGraph` object. The file should have the following format:

- The first line should contain two integers separated by a space, representing the number of vertices and the number of edges, respectively.

- The following `number_of_edges` lines should contain three integers separated by a space, representing the start vertex, the end vertex, and the cost of the edge, respectively.
- The vertices are numbered from 0 to `number_of_vertices - 1`.

Parameters:

- `filename` (str): the name of the file to read the graph from.

Returns:

- A `DirectedGraph` object representing the graph read from the file.

`read_graph_from_file_2(filename: str) -> DirectedGraph`

Reads a directed graph from a text file and returns it as a `DirectedGraph` object. The file should have the following format:

- Each line should contain three integers separated by a space, representing the start vertex, the end vertex, and the cost of the edge, respectively.
- The vertices are numbered from 0 and up, and the file should contain all the edges in the graph.

Parameters:

- `filename` (str): the name of the file to read the graph from.

Returns:

- A `DirectedGraph` object representing the graph read from the file.

`randgraph(n: int, m: int) -> DirectedGraph`

Generates a random directed graph with `n` vertices and `m` edges, where the costs of the edges are random integers between 1 and 100. The graph is returned as a `DirectedGraph` object.

Parameters:

- `n` (int): the number of vertices in the graph.
- `m` (int): the number of edges in the graph.

Returns:

- A `DirectedGraph` object representing the randomly generated graph.

Preconditions and Postconditions for `read_graph_from_file_1(filename: str)`

Preconditions

- `filename` is a string representing the name of a file that exists and can be read by the program.
- The file specified by `filename` has the following format:
  - The first line contains two integers separated by a space, `number_of_vertices` and `number_of_edges`, representing the number of vertices and edges in the graph, respectively.
  - The next `number_of_edges` lines each contain three integers separated by a space, `x`, `y`, and `cost`, representing an edge from vertex `x` to vertex `y` with cost `cost`.
  - The vertices and edges in the file are valid, meaning that there are no duplicate vertices or edges, and all vertices and edges are within the range of valid values for the `DirectedGraph` class.

Postconditions

- Returns a `DirectedGraph` object that represents the graph described in the file specified by `filename`.

Preconditions and Postconditions for `read_graph_from_file_2(filename: str)`

Preconditions

- `filename` is a string representing the name of a file that exists and can be read by the program.
- The file specified by `filename` has the following format:
  - Each line contains three integers separated by a space, `x`, `y`, and `cost`, representing an edge from vertex `x` to vertex `y` with cost `cost`.
  - The vertices and edges in the file are valid, meaning that there are no duplicate vertices or edges, and all vertices and edges are within the range of valid values for the `DirectedGraph` class.

Postconditions

- Returns a `DirectedGraph` object that represents the graph described in the file specified by `filename`.

Preconditions and Postconditions for `random_graph(number_of_vertices: int, number_of_edges: int)`

Preconditions

- **`number_of_vertices`** is a positive integer representing the number of vertices to be generated in the random graph.
- **`number_of_edges`** is a positive integer representing the number of edges to be generated in the random graph. **`number_of_edges`** must be less than or equal to the maximum number of edges that can be generated in a directed graph with **`number_of_vertices`** vertices, which is **`number_of_vertices * (number_of_vertices - 1)`**.

Postconditions

- Returns a **`DirectedGraph`** object with **`number_of_vertices`** vertices and **`number_of_edges`** randomly generated edges, where each edge is assigned a random weight between 1 and 100.