**Graph Algorithms - practical work no. 1**

To address the problem statement, the Python programming language is utilized. A directed graph is represented by a class called *DirectedGraphDictionaryRepresentation,* this class is initialized with three dictionary, first dictionary(*dictionaryIn*) is used to keep track of the inbound vertices of each vertex, with the keys being the vertices themselves, second dictionary(*dictionaryOut*) same as first but for outbound vertices and the third dictionary(*dictionaryCost*) is used to maintain the edges and their associated costs, with the edges serving as the keys, while a *UI* class is defined to test the operations. Additionally, three extra functions are created to read from, write to a file and to create a random graph with specified number of vertices and of edges. An edge is defined as a tuple consisting of an *sourceVertex* and a *targetVertex*.

The *DirectedGraphDictionaryRepresentation* class will offer the subsequent methods:

* *getNumberOfVertices*: returns the number of vertices in the graph.
* *getNumberOfEdges*: returns the number of edges in the graph.
* *getDictionaryIn*: returns a dictionary representing the inbound edges of each vertex in the graph.
* *getDictionaryOut*: returns a dictionary representing the outbound edges of each vertex in the graph.
* *getDictionaryCost*: returns a dictionary representing the costs of each edge in the graph.
* *checkIfTheGraphHasGivenVertex*(*givenVertex*): checks if the given vertex is present in the graph. Returns True if the vertex exists, False otherwise.
* *checkIfTheGraphHasGivenEdge*(*sourceVertex*, *targetVertex*): checks if the given edge (sourceVertex, targetVertex) is present in the graph. Raises a *GraphError* if either the source or target vertex does not exist in the graph. Returns True if the edge exists, False otherwise.
* *addNewVertex*(*vertexToBeAdded*): adds a new vertex to the graph. Raises a *GraphError* if the vertex already exists. Returns True if the vertex was added successfully.
* *addNewEdge*(*sourceVertex*, *targetVertex*, *costOfEdge*): adds a new edge to the graph. Raises a *GraphError* if either the source or target vertex does not exist in the graph, or if the edge already exists. Returns True if the edge was added successfully.
* *parseSetOfVertices*(): yields each vertex in the graph.
* *parseSetOfOutboundEdgesOfAVertex*(*givenVertex*): yields each vertex that is the target of an outbound edge from the given vertex. Raises a *GraphError* if the given vertex does not exist in the graph.
* *parseSetOfInboundEdgesOfAVertex*(*givenVertex*): yields each vertex that is the source of an inbound edge to the given vertex. Raises a *GraphError* if the given vertex does not exist in the graph.
* *getInDegreeOfGivenVertex*(*givenVertex*): returns the in-degree of the given vertex. Raises a *GraphError* if the given vertex does not exist in the graph.
* *getOutDegreeOfGivenVertex*(*givenVertex*): returns the out-degree of the given vertex. Raises a *GraphError* if the given vertex does not exist in the graph.
* *getCostOfGivenEdge*(*sourceVertex*, *targetVertex*): This function returns the cost of a given edge. It takes two parameters *sourceVertex* and *targetVertex* which represent the vertices that form the edge. Raises a *GraphError* if the given edge does not exist in the graph.
* *updateTheCostOfGivenEdge*(*sourceVertex*, *targetVertex*, *newCostOfTheEdge*): This function updates the cost of a given edge. It takes three parameters *sourceVertex* and *targetVertex* which represent the vertices that form the edge, and *newCostOfTheEdge* which represents the new cost of the edge. Raises a *GraphError* if the given edge does not exist in the graph.
* *removeAnEdge*(*sourceVertex*, *targetVertex*): This function removes a given edge from the graph. It takes two parameters *sourceVertex* and *targetVertex* which represent the vertices that form the edge. Raises a *GraphError* if the given edge does not exist in the graph.
* *removeVertex*(*givenVertex*): This function removes a given vertex and all edges that are connected to it from the graph. It takes one parameter *givenVertex* which represents the vertex to be removed. Raises a *GraphError* if the given vertex does not exist in the graph.

The three external functions, which enable reading/writing from/in a file and generating a random graph with a defined number of vertices and edges, are:

* *readGraphFromTextFile*(*textFileName*): This function reads a graph from a text file and returns a new graph object. It takes one parameter *textFileName* which is the name of the text file containing the graph data.
* *writeGraphToTextFile*(*textFileName*, *graph*): This function writes a given graph to a text file, with the file name specified by the *textFileName* argument. The graph is represented using the *DirectedGraphDictionaryRepresentation* class. The function first opens the file in write mode and writes the number of vertices and edges in the graph as the first line. Then, for each edge in the graph, it writes the source vertex, target vertex, and cost of the edge on separate lines. Finally, it writes any vertices that have zero in-degree and out-degree on separate lines.
* *createRandomGraph*(*numberOfVertices*, *numberOfEdges*): This function creates a random directed graph with a given number of vertices and edges. The graph is represented using the *DirectedGraphDictionaryRepresentation* class. The function first creates an empty graph with *numberOfVertices* vertices. It then randomly selects a source vertex and a target vertex and adds an edge between them with a random cost. It repeats this process until the desired number of edges has been added to the graph.

The UI class is initialized with the subsequent data:

* A dictionary(*allCommands*) containing the commands that can be executed in the application. The keys of the dictionary are strings representing the command number, and the values are the corresponding functions that execute the command.
* A list(*listOfGraphs*) that stores the created graphs and is utilized for switching between them. It is initially empty.
* An integer(*indexOfCurrentGraph*) that keeps the index of the current graph that is used by the user.

The *UI* class is designed to provide users with a user-friendly interface to test and utilize the existing methods offered by the *DirectedGraphDictionaryRepresentation* class. Upon starting the program, an empty graph is initialized, and the user is presented with various options to modify it. Additionally, the graph can be stored in an output file for use in future runs of the application. The *UI* class offers an intuitive interface for the user to interact with the graph and perform various operations on it. By presenting the menu options and allowing the user to choose the desired operation, the class ensures that the user has complete control over the graph and can modify it according to their needs.