**Documentation- Practical Work No.1**

This implementation uses the Python language. We shall define a class named *TripleDictionaryGraph* representing a directed graph using three dictionaries. An edge is represented as a tuple(origin and target). The class mentioned above will give us access to the following methods:

* *def dictionaryCost(self)* – returns the cost of an edge
* *def dictionaryIn(self)* - returns a dictionary of inbound vertices
* *def dictionaryOut(self)* – returns a dictionary of outbound vertices
* *def numberOfVertices(self)* – returns the number of vertices
* *def numberOfEdges(self)* - returns the number of edges
* *def parseVertices(self)* – it is an iterator for the vertices
* *def parseInBound(self,x)* – it is an iterator for the inbound vertices of x, that is a vertex
* *def parseOutBound(self,x)* – it is an iterator for the outbound vertices of x, that is a vertex
* *def parseCost(self,x)*  - it is an iterator for the edges of the graph and their costs
* *def addVertex(self, x) -* adds a new vertex x to the graph, the vertex shall not already be in the dictionary, returns False if it doesn’t add it and True otherwise
* *def removeVertex(self, x) -* removes a vertex x from the graph, the vertex x must already be in the dictionary, returns False if it doesn’t remove it and True otherwise
* *def inDegree(self, x) -* returns the in degree of the vertex x or -1 if the vertex does not exist
* *def outDegree(self, x) -* returns the out degree of a vertex x or -1 if the vertex does not exist
* *def addEdge(self, x, y, cost) -* adds a new edge (x, y) to the graph, that is not already in the graph, returns False if it doesn’t add and True otherwise
* *def removeEdge(self, x, y) -* removes an edge (x, y) from the graph, that already is in the graph, returns False if it doesn’t remove it and True otherwise
* *def findIfEdge(self, x, y) -* returns the cost of the edge (x, y) if it exists and False otherwise
* *def changeCost(self, x, y, cost) -* changes the cost of an edge (x, y) to be equal to cost, returns True if it changes, False otherwise
* *def makeCopy(self) -* return a copy of the graph

Reading/Writing functions from/in a file are:

* *def writeGraphToFile(graph, file)* – it receives as parameters a graph(that must not be empty) and a file and it writes in a file the graph, if the dictionaries used for printing are empty a ValueError will be raised
* *def readGraphFromFile(filename) -* it has as a parameter a file name from which the graph will be read and returns the current graph that has been read

The UI class will give us access to the following methods:

* *def addEmptyGraph(self) –* adds an empty graph to the list of graphs
* *def createRandomGraphUi(self) –* it receives as input a number of vertices and edges; adds that graph to the list of graphs
* *def generateRandom(self, vertices, edges) –* it is a static method – it is used by the function createRandomGraphUi that creates a random graph
* *def readGraphFromFileUi(self) –* it reads from a file that is given by the user a graph that it will be added to the list of graphs
* *def writeGraphToFileUi(self) –* it writes the current graph
* *def switchGraphUi(self) –* it switches the graphs given as an input by the user
* *def getNumberOfVerticesUi(self) –* prints the number of vertices
* *def getNumberOfEdgesUi(self) –* prints the number of edges
* *def listOutbound(self) –* lists all the vertices from the current graph with all their outbound vertices
* *def listInbound(self) –* lists all the vertices from the current graph with all their
* inbound vertices
* *def listAllCosts(self) –* lists all the edges with their cost
* *def parseAllVertices(self) –* lists all the vertices
* *def addVertexUi(self) –* it receives an input for the vertex and uses the addVertex method from the class *TripleDictionaryGraph*, alsoprinting a corresponding message
* *def deleteVertexUi(self) –* it receives an input for the vertex and uses the removeVertex method from the class *TripleDictionaryGraph,* alsoprinting a corresponding message
* *def addEdgeUi(self) –* it receives an input for vertices and cost and uses the addEdge method from the class TripleDictionaryGraph, also printing a corresponding
* *def removeEdgeUi(self) -* it receives an input for vertices and uses the removeEdge method from the class *TripleDictionaryGraph,* alsoprinting a corresponding
* *def modifyCostUi(self) –* it receives an input for vertices and cost and uses the changeCost method from the class *TripleDictionaryGraph,* alsoprinting a corresponding
* *def getInDegreeUi(self) –* prints the in degree of a vertex received as an input
* *def getOutDegreeUi(self) –* prints the out degree of a vertex received as an input
* *def checkIfEdgeUi(self) -* checks if an edge given by the user exists and it prints its cost
* *def copyCurrentGraphUi(self) –* creates a copy of the graph using the makeCopy method from the class *TripleDictionaryGraph*
* *def printMenu()*  - static method – prints the directed graph menu to help the user
* *def start(self)* – starts the program

The main module is calling the start function so the program can run.