**Practical work no. 1**

**Specification:**

We shall define two classes. One called Dictionaries which will be represented by three dictionaries and the number of vertices and the number of edges. The other one is called Console, which will represent the user interface console.

**The class Dictionaries will provide the following methods:**

**def outboundEdges(self,vertex):**

**"""**

**returns the outbound edges of a vertex**

**vertex - integer**

**"""**

**def inboundEdges(self,vertex):**

**"""**

**returns the outbound edges of a vertex**

**vertex - integer**

**"""**

**def read\_file(self,file\_name):**

**"""**

**Reads a graph from a file**

**file\_name - the name of the file (string)**

**"""**

**def removeVertex(self,vertex):**

**"""**

**removes a vertex**

**vertex - integer**

**"""**

**def getCost(self,vertex1,vertex2):**

**"""**

**returns the cost of an edge**

**vertex1, vertex2 - integers**

**"""**

**def modifyCost(self,vertex1,vertex2,newCost):**

**"""**

**modifies the cost of an edge**

**vertex1, vertex2 integers**

**newCost - new cost, integer**

**"""**

**def isEdge(self,vertex1,vertex2):**

**"""**

**return True if there is an edge between the 2 vertices, False otherwise**

**"""**

**def inDegree(self,vertex):**

**"""**

**returns the in degree of a vertex**

**vertex - integer**

**"""**

**def outDegree(self,vertex):**

**"""**

**returns the out degree of a vertex**

**vertex - integer**

**"""**

**def addEdge(self,vertex1,vertex2,cost):**

**"""**

**adds an edge between vertex1 and vertex2 with the cost givem**

**vertex1 - first vertex, integer**

**vertex2 - second vertex, integer**

**cost - cost, integer**

**"""**

**def removeEdge(self,vertex1,vertex2):**

**"""**

**Removes an edge**

**or raise ValueError if it doesn't exist**

**"""**

**def addVertex(self):**

**"""**

**adds a vertex to the graph**

**"""**

**def nrVertices(self):**

**"""**

**returns the number of vertices**

**"""**

**def vertices(self):**

**"""**

**returns the list of vertices**

**"""**

**def copyGraph(self,newGraph):**

**"""**

**copies the graph**

**new\_graph - a new copy of the graph**

**"""**

**The class Console will provide the following methods:**

**def menu(self):**

**"""**

**prints the commands from which the user chooses the options**

**"""**

**def run\_menu(self):**

**"""**

**starts the execution of the program**

**"""**

**There are a few other external functions:**

**def writeGraphExternally(graph,file\_name):**

**"""**

**writes the graph in a file**

**graph - the graph (corresponding to the Dictionaries class)**

**file\_name - name of the file (string)**

**"""**

**def readTheGraphExternally(graph,file\_name):**

**"""**

**reads the graph from a file**

**graph - the graph (corresponding to the Dictionaries class)**

**file\_name - name of the file (string)**

**"""**

**def createRandomGraph(number\_of\_vertices,number\_of\_edges):**

**"""**

**creates a random graph**

**number\_of\_vertices - integer**

**number\_of\_edges - integer**

**"""**

**# exception cases**

**class CompleteGraphException(Exception):**

**"""**

**Raises an exception if the complete graph condition is fulfilled**

**"""**

**class ItAlreadyExists(Exception):**

**"""**

**Raises an exception if the edge already exists**

**"""**

**class EdgeHasNotBeenFound(Exception):**

**"""**

**Raises an exception if the edge has not been found**

**"""**

**The class Console will have the following data members:**

**self.graph = Dictionaries()**

**self.graph.read\_file(file\_name)**

**The class Dictionaries will have the following data members:**

**self.n = 0 # number of vertices**

**self.m = 0 # number of edges**

**self.D\_out = {} # dictionary of outbound vertices**

**self.D\_in = {} # dictionary of inbound vertices**

**self.D\_cost = {} # dictionary of edges costs**