**Practical Work no.1**

**Danicico George-Iulian Group 912**

**Specifications:**

We will have 2 classes. The first class, DirectedGraph will represent a directed weighted graph. The second class, UI will provide access to the first class functionalities. The Graph will be represented using 3 list of dictionaries, one to represent the inbound neighbours for each vertex, one to represent the outbound neighbours for each vertex and one to represent the cost for every edge.

Each vertex is uniquely identified by an integer number which represents its index. Each edge is uniquely identified by a tuple formed of 2 integer numbers, representing the indexes of the vertices that the edge unites.

The class DirectedGraph provides the following public methods:

* + **NrOfEdges =>** **getter** that returns the number of edges.
  + **NrOfVertices =>** **getter** that returns the number of vertices.
  + **NrOfEdges(number) =>** changes the number of edges.
  + **NrOfVerticesEdges(number)** => changes the number of vertices.
  + **getAllVertices() =>** returns all the vertices in the graph.
  + **getGraphCopy() =>** returns a copy of the current graph.
  + **getInboundNeighbours(vertex) =>** returns a list of all inbound neighbours for a vertex.
  + **getOutboundNeighbours(vertex) =>** returns a list of all Outbound neighbours for a vertex.
  + **getCostOfEdge(edge) =>** returns the cost for an edge which is identified as a tuple of 2 integer numbers.
  + **getInDegreeVertex(vertex) =>** returns the indegree for a vertex.
  + **getOutDegreeVertex(vertex) =>** returns the outdegree for a vertex.
  + **setCostEdge(edge, cost) =>** sets a new cost for the edge.
  + **vertexExistence(vertex) =>** checks if a vertex exists in the graph.
  + **edgeExistence(edge) =>** checks if a given edge exists in the graph.
  + **deleteVertex(vertex) =>** deletes a given vertex from the graph.
  + **deleteEdge(edge) =>** deletes a given edge from the graph.
  + **addEdge(edge) =>** adds a new edge in the graph.
  + **addVertex(vertex)** => adds a new vertex in the graph.
  + **generateRandomGraph(vertices, edges)** => generates a random graph.
  + **loadGraph1, loadGraph2, writeGraph1, writeGraph2** are methods that read from a file and print to a file a graph in 2 different formats. They all have a common parameter, **file\_name**, which represents the path to the file.

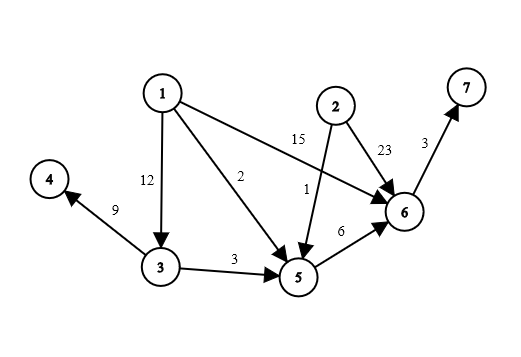
**Implementation:**

The class DirectedGraph provides the following private methods and fields:

* + **\_\_dictIn =>** Dictionary that keeps the inbound neighbours for every vertex.
  + **\_\_dictOut =>** Dictionary that keeps the inbound neighbours for every vertex.
  + **\_\_dictCost=>** Dictionary that keeps the cost for every edge.
  + **\_\_numberOfEdges =>** integer variable that keeps the total number of edges.
  + **\_\_numberOfVertices =>** integer variable that keeps the total number of vertices.
  + **\_initGraph() =>** a method that initialise an empty graph.

The class UI processes the data the user enters, accesses the DirectedGraph functionalities, and prints to the screen what the user desires.

Below there is a representation that illustrates how the 3 dictionaries would look like.



DictIn = { DictOut = { DictCost = {

1 : [] 1 : [3, 5, 6] (1, 3) : 12 (5, 6) : 6

2 : [] 2 : [5, 6] (1, 5) : 2 (6, 7) : 3

3 : [1] 3 : [4, 5] (1, 6) : 15 }

4 : [3] 4 : [] (2, 5) : 1

5 : [1, 2, 3] 5 : [6] (2, 6): 23

6 : [1, 2, 5] 6 : [7] (3, 4) : 9

7 : [6] 7 : [] (3, 5) : 3

} }