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 graph 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.
  The complexity is O(1).
- **NrOfVertices => getter** that returns the number of vertices.
  The complexity is O(1).
- **NrOfEdges(number) => setter** that changes the number of edges.
  The complexity is O(1).
- **NrOfVerticesEdges(number)** =>**setter** that changes the number of vertices.
  The complexity is O(1).
- **getAllVertices() => returns** all the vertices in the graph. No preconditions or postconditions. The complexity is O(n).
- **getGraphCopy() => returns** a copy of the current graph. No preconditions or postconditions. The complexity is O(1).
- **getInboundNeighbours(vertex) => returns** a list of all inbound neighbours for a vertex. **Precoditions**: the vertex exists.
  Postconditions: -. The complexity is O(1).
- **getOutboundNeighbours(vertex) => returns** a list of all Outbound neighbours for a vertex. **Preconditions:** the vertex exists.
  Postconditions: -. The complexity is O(1).
- **getCostOfEdge(edge) => returns** the cost for an edge which is identified as a tuple of 2 integer numbers.

Preconditions: the edge exists.
Postconditions: -. The complexity is O(1).

- **getInDegreeVertex(vertex) => returns** the indegree of a vertex.
  Preconditions: the vertex exists.
  Postconditions: -. The complexity is O(1).
- **getOutDegreeVertex(vertex) => returns** the outdegree of a vertex.
  Preconditions: the vertex exists.
  Postconditions: -. The complexity is O(1).
- **setCostEdge(edge, cost) =>** sets a new cost for the edge.
  Preconditions: the edge exists and the cost is a valid integer.
  Postconditions: the edge has a new cost. The complexity is O(1).
- **vertexExistence(vertex) =>** checks if a vertex exists in the graph.
  Preconditions: the vertex is a valid integer.
  Postconditions: -. **Returns** True if the vertex is in the graph/False otherwise. The complexity is O(1).
- **edgeExistence(edge) =>** checks if a given edge exists in the graph.
  Preconditions: the edge is a valid tuple of 2 integers.
  Postconditions: - . **Returns** True if the edge exists, false otherwise. The complexity is O(1).
- **deleteVertex(vertex) =>** deletes a given vertex from the graph, if it exists.
  Preconditions: the vertex exists.
  Postconditions: the vertex has been deleted from the graph. The complexity is O(n).
- **deleteEdge(edge) =>** deletes a given edge from the graph.
  Preconditions: the edge exists.
  Postconditions: the edge has been deleted from the graph. The complexity is O(1).
- **addEdge(edge) =>** adds a new edge in the graph.
  Preconditions: the edge does not exists and is a valid tuple of 2 integers.
  Postconditions: the edge has been added to the graph .The complexity is O(1).
- **addVertex(vertex)** => adds a new vertex in the graph.
  Preconditions: the vertex does not exists and is a valid integer.
  Postconditions: the vertex has been added to the graph.The complexity is O(1).
- **generateRandomGraph(vertices, edges)** => generates a random graph.
  Preconditions: vertices and edges are valid integer numbers.
  Postconditions: a random graph with the given number of vertices and edges.
- **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.

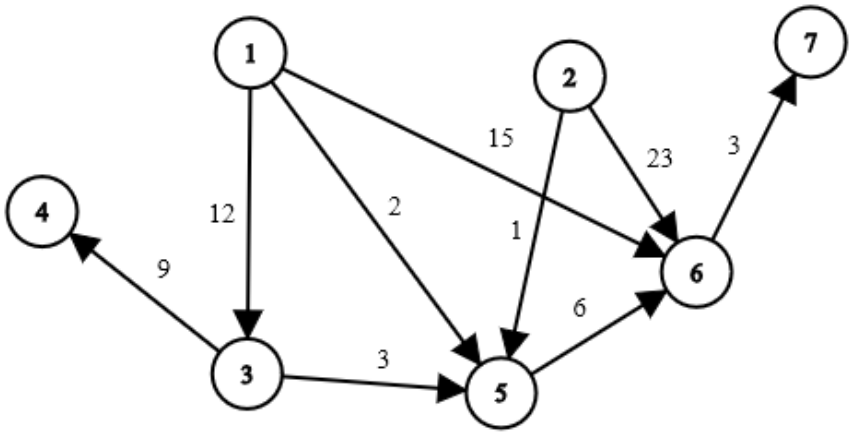
Implementation:

The class DirectedGraph provides the following private methods and fields:

- **__dictIn => ** Dictionary that keeps the inbound neighbours for every vertex. Each key will be the represented by a vertex from the graph, and its corresponding value will be a list that contains the inbound neighbours.
- **__dictOut =>** Dictionary that keeps the inbound neighbours for every vertex.
- Each key will be the represented by a vertex from the graph, and its corresponding value will be a list that contains the outbound neighbours.
- **__dictCost=>** Dictionary that keeps the cost for every edge. Each key will be represented as a tuple of 2 vertices representing the ending points of an edge. The value will be the cost of the 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 = {

1 : []

2 : []

3 : [1]

4 : [3]

5 : [1, 2, 3]

6 : [1, 2, 5]

7 : [6]

}

DictOut = {

1 : [3, 5, 6]

2 : [5, 6]

3 : [4, 5]

4 : []

5 : [6]

6 : [7]

7 : []

}

DictCost = {

(1, 3) : 12

(1, 5) : 2

(1, 6) : 15

(2, 5) : 1

(2, 6): 23

(3, 4) : 9

(3, 5) : 3

(5, 6) : 6

(6, 7) : 3

}