**Documentation PRACTICAL WORK 1**

In domain.directed\_graph I have created the class DirectedGraph.

It’s \_\_init\_\_ has the following attributes:

\_\_dictIn – a dictionary that will have as keys vertices, and as values a list of vertices; key-value represents an edge; the predecessor is the key and the successors are stored in that list

\_\_dictOut – also a dictionary, the opposite of dictIn; every key represents a vertex successor and the value is a list of predecessors for that vertex

\_\_costs – a dictionary that has edges(tuples) as keys, and their value is the cost

Functions:

Getters: no parameters

dictIn : it returns self.\_\_dictIn

analogy for dictOut, costs

isVertex:

Checks if a vertex given as a parameter exists or not.

* Parameters: vertexToCheck – integer, represent the vertex which existance is checked
* It returns a bool, true if the vertex exists, false otherwise

addVertex:

Adds a vertex to the graph, by adding the key vertexToAdd to \_\_dictIn and \_\_dictOut if the vertex does not exist already.

* Parameters: vertexToAdd – integer, the vertex that will be added
* Return: none

removeVertex:

Removes a vertex from the graph, if it exists, by deleting the keys vertexToRemove from \_\_dictIn and \_\_dictOut, also removing all edges corresponding to it.

For every vertex, vertexToRemove is removed from their \_\_dictIn / \_\_dictOut, if it’s in the value – list . Also all edges that contain it are removed from \_\_costs.

* vertexToRemove – integer, the vertex that will be removed
* return: none

isEdge:

Checks if a tuple, pair of its, is an edge or not.

* Parameters: edgeToCheck – tuple, representing an edge
* Return: true if edgeToCheck is and edge, false otherwise

addEdge:

Adds an edge to the graph, if it does not already exist.

It will add to the list value at \_\_dictIn(edgeToAdd[0]), edgeToAdd[1]. Analogy for \_\_dictOut. It will also add the key edgeToAdd in self.\_\_costs with the value cost.

* Parameters: edgeToAdd – tuple, representing the edge we want to add
* Cost – the cost of that edge
* Return: none

removeEdge:

getNumberOfVertices:

Gets the number of vertices ( by returning the number of keys in dictIn).

* Parameters: none
* Return: int, the number of vertices in the graph

getSetOfVertices:

Gets the set of all vertices ( by returning all the keys in dictIn)

* Parameters: none
* Return: list, all the vertices in the graph

getInDegree:

Returns the in degree of a given vertex(by returning the length of the list at dictIn[vertex]).

* Parameters: vertex – int, the vertex which indegree we want to get
* Return: int, the indegree of the given vertex

getOutDegree:

Analogy getInDegree, but with dictOut

getInBoundEdges:

Gets all the inBoundEdges of a given vertex, if that vertex exists.

* Parameters: vertex – int, the vertex whose inBoundEdges we want to get
* Return: list, representing all the edges that have the given vertex as a starting point ( as a predecessor)

getOutBoundEdges:

Analogy for getInBoundEdges, but this time the given vertex is the successor, the end vertex.

getAllEdges:

Returns a list of all the edges.

* Parameters: none
* Return: list; a list of tuples, each tuples representing an edge in the graph

createCopy:

Creates a copy of the current graph and returns in.

* Parameters: none
* Return: DirectedGraph

getCost:

Gets the cost of a certain edge, if that edge exists.

* Parameters: edge – tuple, the edge which costs we want to get
* Return: int, the cost of the given edge

setCost:

Sets the cost of a certain edge, if that edge exists.

-Parameters: edge- tuple, the edge whose cost we want to change

Cost – int, the new cost

* Return: none

readGraphFromFileStandard:

Reads a graph from a file with standard format:

NumberOfVertices numberOfEdges

Vertex vertex cost

Vertex vertex cost

Vertex vertex cost

Vertex vertex cost

…

* Parameters: filename – str, the name of the file we are reading from
* Return: DirectedGraph, the graph we’ve read from the given file

readGraphFromFile2:

Reads a graph from a file that we’ve written, that has the following format:

Vertex vertex cost

Vertex vertex cost

…

* Parameters: filename -str, the name of the file we are reading from
* Return: DirectedGraph

writeGraphToFile:

Writes a graph to a given file, with the following format

Vertex vertex cost

Vertex vertex cost

…

* Parameters: filename – str, the name of the file we are writing into
* Return: none