Documentation for practical work no. 1

# The Graph class:

Attributes:

\_vertexes (int; keeps the number of vertexes in the graph)  
 \_vertex\_dict\_out (dict; keeps on every key the list of outbound neighbours of the vertex represented by that key)  
 \_vertex\_dict\_in (dict; keeps on every key the list of inbound neighbours of the vertex represented by that key)

\_edge\_dict (dict; keeps on every key (tuple representing the source vertex and the destination vertex of the edge) the value/cost of the edge represented by that key)

def \_\_init\_\_(self, vertexes=0):  
 *"""  
 Generates a graph with the given number of vertexes* ***:param*** *vertexes: the number odd vertexes (int)* ***:raises*** *ValueError: if the number of vertexes given is < 0  
 """*

def parseX(self):  
 *"""  
 Returns a list with all the vertexes in the graph* ***:return****: a list with all the vertexes in the graph (list of int)  
 """*

def parseNOut(self, x):  
 *"""  
 Returns all the outbound neighbours of the given vertex x as a list* ***:param*** *x: the given vertex (int)* ***:return****: a list of all the outbound neighbours of x (list of int)  
 """*

def parseNIn(self, x):  
 *"""  
 Returns all the inbound neighbours of the given vertex x as a list* ***:param*** *x: the given vertex (int)* ***:return****: a list of all the inbound neighbours of x (list of int)  
 """*

def add\_edge(self, x, y, val=0):  
 *"""  
 Adds an edge to the graph between the given coordinates and with a given value* ***:param*** *x: the source vertex (int)* ***:param*** *y: the destination vertex (int)* ***:param*** *val: the cost/information of the edge (int)* ***:return****: True if the edge can be added, False if the edge already exists  
 preconditions: the edge (x, y) must not exist in the dictionary of edges and the vertices x and y have to exist  
 post operation: the new edge (x, y) has been added to the dictionary of edges  
 """*

def remove\_edge(self, x, y):  
 *"""  
 Removes the edge between the given vertexes* ***:param*** *x: the source vertex (int)* ***:param*** *y: the destination vertex (int)* ***:return****: True if the edge can be removed, False if the edge does not exist  
 preconditions: the edge (x, y) must exist in the dictionary of edges  
 post operation: the edge (x, y) no longer exists in the dictionary of edges; vertex x is removed from the inbound neighbour dict of y and y is removed from the outbound neighbour dictionary of x  
 """*

def add\_vertex(self, x):  
 *"""  
 Adds the given vertex to the graph* ***:param*** *x: the given vertex (int)* ***:return****: True if the vertex has been added, False if the vertex already exists  
 precondition: the vertex x must not exist in either of the outbound/inbound neighbours dictionaries  
 post operation: the vertex x is added as a key in the outbound/inbound neighbours dictionaries and the number of vertexes in the graph is increased  
 """*

def remove\_vertex(self, x):  
 *"""  
 Removes a given vertex from the graph* ***:param*** *x: the given vertex (int)* ***:return****: True if the vertex has been removed, False if the vertex does not exist  
 preconditions: the vertex x must exist as a key in both the outbound/inbound neighbours dictionaries  
 post operation: the vertex x is removed from both the outbound/inbound neighbours dictionaries and every edge associated with the vertex is removed; the number of vertexes in the graph is decreased  
 """*

def get\_info(self, x, y):  
 *"""  
 Returns the cost/info stored on the edge given by the vertexes x and y* ***:param*** *x: the source vertex (int)* ***:param*** *y: the destination vertex (int)* ***:return****: The information if the edge exists, None otherwise  
 preconditions: the edge (x, y) must exist in the dictionary of edges  
 """*

def set\_edge(self, x, y, val):  
 *"""  
 Modifies the cost/info at the given edge* ***:param*** *x: the source vertex (int)* ***:param*** *y: the destination vertex (int)* ***:param*** *val: the new cost/information of the edge (int)* ***:return****: True if the cost/information has been changed, False if the edge does not exist  
 preconditions: the edge (x, y) must exist in the dictionary of edges  
 """*

def in\_degree(self, x):  
 *"""  
 Returns the in degree of a given vertex* ***:param*** *x: the given vertex (int)* ***:return****: the in degree of the given vertex if it exists, None otherwise  
 preconditions: the vertex x must exist as a key in the inbound neighbours dictionary  
 """*

def out\_degree(self, x):  
 *"""  
 Returns the out degree of a given vertex* ***:param*** *x: the given vertex (int)* ***:return****: the out degree of the given vertex if it exists, None otherwise  
 preconditions: the vertex x must exist as a key in the outbound neighbours dictionary  
 """*

def check\_edge(self, x, y):  
 *"""  
 Checks if the given edge is in the graph* ***:param*** *x: the source vertex (int)* ***:param*** *y: the destination vertex (int)* ***:return****: True if it is in the graph, False otherwise  
 """*

def check\_vertex(self, x):  
 *"""  
 Checks if the given vertex is in the graph* ***:param*** *x: the given vertex (int)* ***:return****: True if it is in the graph, False otherwise  
 """*

def copy\_graph(self):  
 *"""  
 Creates a copy of the current graph which does not modify it* ***:return****: a copy of the current graph  
 """*

def get\_vertex\_count(self):  
 *"""  
 Returns the number of vertexes in the graph* ***:return****: the number of vertexes in the graph (int)  
 """*

def get\_edge\_count(self):  
 *"""  
 Returns the number of edges in the graph* ***:return****: the number of edges in the graph (int)  
 """*