**Graph Algorithms- practical work no1**

**Documentation for the implementation in python**

We have a new class: *Graph*

Fields:

* numberOfVertices – the number of vertices
* dictIn – the dictionary containing for each vertex the list of inbound neighbours
* dictOut – the dictionary containing for each vertex the list of outbound neighbours
* dictCosts – the dictionary containing each cost for each edge

Functions:

* **def get\_numberOfVertices()**
* returns an integer containing the number of vertices in the directed graph
* **def parseVertices()**
* returns a copy of all the vertices in the directed graph
* **def isEdge(x,y)**
* returns true if there is an edge from x to y, false otherwise
* **def getInDegree(x)**
* returns an integer representing the in degree of the vertex x
* precondition: x needs to be a valid vertex in the graph, in case it isn’t, the error is handled
* **def getOutDegree(x)**
* returns an integer representing the out degree of the vertex x
* precondition: x needs to be a valid vertex in the graph, in case it isn’t, the error is handled
* **def parseIterableIn(x)**
* returns a copy of all “in” neighbours of vertex x
* **def parseIterableOut(x)**
* returns a copy of all “out” neighbours of vertex x
* **def addEdge(start,end,cost)**
* adds an edge (it has a start and an end) that has a cost
* precondition: the edge mustn’t already exist and the vertices need to be valid, otherwise the error is handled
* **def removeEdge(x,y)**
* removes the edge (x,y) from the graph
* precondition: (x,y) needs to be a valid edge in the graph, otherwise the error is handled
* **def retrieveCost(start,end)**
* returns the cost of (start,end) edge
* precondition: (x,y) needs to be a valid edge in the graph, otherwise the error is handled
* **def addVertex(x)**
* adds the vertex x to the graph, as an isolated vertex
* precondition: x mustn’t already be a vertex in the graph, if it is, the error is handled
* **def removeVertex(x)**
* removes the vertex x from the graph
* precondition: x needs to exist as a vertex in the graph, otherwise the error is handled
* **def copyGraph()**
* returns a copy of the graph
* **def readGraph\_fromFile(filename)**
* returns graph read from file
* precondition: the file must exist
* **def writeGraph\_fromFile(filename,graph)**
* writes graph to a file
* **def createRandomGraph(numberOfVertices,numberOfEdges)**
* returns a graph created randomly with the number of vertices and the number of edges
* **def allIsolatedVertices()**
* returns an iterable containing all the isolated vertices
* **def modifyEdgeCost(start,end,newCost)**
* modifies the cost of an edge(start,end)
* precondition: (start,end) must be a valid edge in the graph, if it isn’t, the error is handled
* **def edges()**
* returns an interable containing all the edges
* **def costs()**
* returns an interable containing all the costs