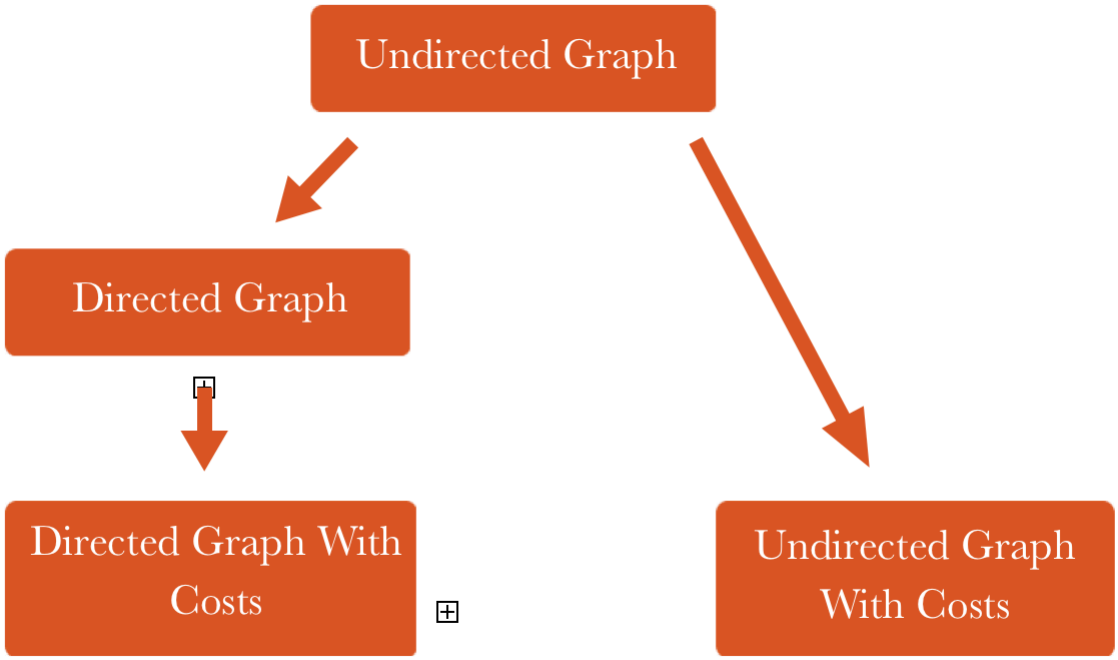
UMircea Sorin-Sebastian
Group 915/1

Practical work nr. 1

Documentation

The philosophy behind my implementation is composed by a series of classes, one derived from another that characterise multiple graph types:

- Undirected graphs
- Directed graphs
- Undirected graphs with cost
- Directed graphs with cost



Graph methods:

- __init__
- parseNodeOut
- isEdge
- addEdge
- addVertex
- getNumberOfVertices
- getNumberOfEdges
- getOutDegree
- removeEdge
- removeVertex
- getGraph

These methods are present in the base class (Undirected Graph) and some of them are overridden by the derived class in order to adapt the functionality accordingly.

Also Directed/Undirected cost graphs classes have methods for changing and getting the cost.

User has access to the following commands for managing the graph:

```
########################################
0. Print graph
1. Load graph from file (graph.txt)
2. Save graph to file (graph.txt)
3. Add edge (muchie)
4. Add vertex (nod)
5. Remove edge (muchie)
6. Remove vertex (nod)
7. Get number of Vertices
8. Get number of EDGES
9. Get cost of EDGE
10. Check if edge exists (between two vertices)
11. In & Out degree of an edge
12. Outbound edges of a vertex
13. Inbound edges of a vertex
14. Modify edge information (integer)
########################################
```

Some code snapshots

UndirectedGraph class:

```
def parseNodeOut(self, node):
    '''
    :param node: integer
    :return: a list with all the successors of the node
    '''
    return self.dictOut[node]

def isEdge(self, x, y):
    '''
    :param x: integer
    :param y: integer
    :return:  True if the edge (x,y) exists, False othersie
    '''

    #check if the edges actually exist
    if not x in self.dictOut.keys() or not y in self.dictOut.keys():
        return False

    return y in self.dictOut[x]

def addEdge(self, x, y):
    '''
    Adds the edge (x,y)
    :param x: integer
    :param y: integer
    '''
    #If the vertex is not create it, do it now
    self.addVertex(x)
    self.addVertex(y)

    if self.isEdge(x, y):
        print(x, y)
        raise graphException("Edge already exists")

    self.dictOut[x].append(y)
    self.dictOut[y].append(x)

def addVertex(self, x):
    if not x in self.dictOut.keys():
        self.dictOut[x] = []

def getNumberOfVertices(self):
    '''
    :return: The number of vertices in the graph (NODURI)
    '''
    self._edges = len(self.dictOut)
    return self._edges
```

DirectedCostGraph class:

```
class DirectedCostGraph(DirectedGraph):
    def __init__(self, file):
        '''
            Creates an directed cost graph with n vertices (noduri) - numbered from 0 to n-
            :param n: integer, number of vertices
        '''
        super().__init__(file)
        self.dictCost = {}

    def addEdge(self, x, y, cost):
        '''
        :param x: vertex
        :param y: vertex
        :param cost: the cost
        :return: adds an edge to the graph
        '''
        super().addEdge(x, y)
        self.dictCost[(x,y)] = cost

    def getCost(self, x, y):
        '''
        :param x: vertex
        :param y: vertex
        :return: cost of edge x-y
        '''
        return self.dictCost[(x,y)]

    def changeCost(self, x, y, cost):
        '''
        :param x: vertex
        :param y: vertex
        :param cost: changes the cost of edge x-y with this value
        :return:
        '''
        if not self.isEdge(x, y):
            raise graphException("Edge does not exist")

        self.dictCost[(x,y)] = cost

    def __str__(self):
        '''
        :return: a string with the graph ready to be saved on file
        '''
        res = ''

        res += str(self.getNumberOfVertices()) + ' ' + str(self.getNumberOfEdges()) + '\n'
        for node in self.dictOut:
            # If a node is isolated
            if len(self.parseNodeOut(node)) == 0 and (len(self.parseNodeIn(node))) == 0:
                res += str(node) + ' -1\n'
                continue

            for j in self.parseNodeOut(node):
                res += str(node) + ' ' + str(j) + ' ' + str(self.getCost(node, j)) + '\n'

        return res
```