**def** \_\_init\_\_(self, file\_name):  
 self.ins={}  
 f = open(file\_name, **"r"**)  
 lines = f.read().split(**"\n"**)  
 self.vertices=int(lines[0].split()[0])  
 **for** i **in** range(self.vertices):  
 self.ins[i] = []  
 self.read\_graph(file\_name)  
  
**def** read\_graph(self, file\_name):  
 *#reands a graph from the file  
 #preconditions: file\_name - name.txt, from where it takes all the information about the graph* f = open(file\_name, **"r"**)  
 lines = f.readline().strip()  
 **while** lines != **""**:  
 line = lines.split(**" "**)  
 x = int(line[0])  
 y = int(line[1])  
 self.ins[y].append(x)  
 self.ins[x].append(y)  
 lines = f.readline().strip()  
  
**def** add\_edge(self,x,y):  
 *# adds an edge to the graph, if this doesn't already exist. if it exists, it exits the function  
 # preconditions: x,y - an integers, 2 vertexes and c - integer, the cost* **if** self.exist\_edge(x,y):  
 **return False  
 if** self.exist\_vertex(x)==**False or** self.exist\_vertex(y)==**False**:  
 **return False** self.ins[y].append(x)  
 **return True  
  
def** parse(self):  
 *#parses the graph and returns all the vertexes* **return** list(self.ins.keys())  
  
**def** print(self):  
 *#prints the graph* **for** i **in** self.parse():  
 **if** len(self.ins[i]) == 0 **and** len(self.ins[i]) == 0:  
 print(i, **"is an isolated vertex"**)  
 **else**:  
 **for** j **in** self.ins[i]:  
 **if** self.exist\_edge(i, j):  
 print(i, **"->"**, j)  
  
**def** get\_number(self):  
 *#returns the number of vertexis in the graph* **return** len(self.ins.keys())  
  
**def** exist\_vertex(self,x):  
 *# checks if a vertex exists. returns True if it exists and False if not  
 # preconditions: x - integer, representing a vertex* **if** x **in** self.parse():  
 **return True  
 return False  
  
def** exist\_edge(self,x,y):  
 *# checks if an edge exists. returns True if it exists and False if not  
 # preconditions: x,y - integers, representing 2 vertexis, x being the begining of the edge, and y the final* **if** self.exist\_vertex(x) **and** self.exist\_vertex(y):  
 **for** i **in** self.ins[x]:  
 **if** i==y:  
 **return True  
 return False  
 return False  
  
  
def** parse\_in(self,x):  
 *# parses the inbound edges of a vertex, and returns the list of the vertexes that go in, if this already exists. if not, it exists the function  
 # preconditions: x - an integer, a vertex* **if** self.exist\_vertex(x)==**False**:  
 **return False  
 return** self.ins[x]  
  
**def** BFS(self, temp,s,visited):  
 q = [s]  
 **while** q:  
 v = q.pop(0)  
 **if not** v **in** temp:  
 temp = temp + [v]  
 q = q + self.ins[v]  
 temp.sort()  
 **return** temp  
  
**def** connectedComponents(self):  
 visited = []  
 cc = []  
 **for** i **in** range(self.get\_number()):  
 visited.append(**False**)  
 **for** v **in** range(self.get\_number()):  
 **if** visited[v] == **False**:  
 temp = []  
 aux=self.BFS(temp, v, visited)  
 **if** aux **not in** cc:  
 cc.append(aux)  
 **return** cc

Vladarean Maria-Monica

917

**Documentation**

I defined a class named Graph representing an undirected graph. This class provides the following methods: