Vladarean Maria-Monica

917

**Documentation**

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

**def** \_\_init\_\_(self, file\_name):

**#defines a graph by 3 dictionaries: one marking the inbound vertexes, one for the outbound vertexes and one for the cost of every edge. so, every dictionary has keys (a vertex, or a pair of vertexes) and a value (a vertex, or a cost)**

self.ins={}  
 self.outs={}  
 self.costs={}  
  
 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.outs[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()  
 lines = f.readline().strip()  
 **while** lines != **""**:  
 line = lines.split(**" "**)  
 x = int(line[0])  
 y = int(line[1])  
 cost = int(line[2])  
  
 self.ins[y].append(x)  
 self.outs[x].append(y)  
 self.costs[(x, y)] = cost  
 lines = f.readline().strip()  
  
**def** add\_edge(self,x,y,c):  
 **# 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)  
 self.outs[x].append(y)  
 self.costs[(x,y)]=c  
 **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.outs[i]) == 0:  
 print(i, **"is an isolated vertex"**)  
 **else**:  
 **for** j **in** self.outs[i]:  
 **if** self.exist\_edge(i, j):  
 print(i, **"->"**, j, **"cost:"**, self.costs[(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.outs[x]:  
 **if** i==y:  
 **return True  
 return False  
 return False  
  
def** get\_cost(self,x,y):  
 **# gets the cost for an edge, if this already exists. if not, it exits the function  
 # preconditions: x,y - integers, representing 2 vertexis, x being the begining of the edge, and y the final**  
 **if**(self.exist\_edge(x,y)):  
 **return** self.costs[(x,y)]  
 **return False  
  
def** set\_cost(self,x,y,cost):  
 **# sets the cost for an edge, if this already exists. if not, it exits the function  
 # preconditions: x,y - integers, representing 2 vertexis, x being the begining of the edge, and y the final**  
 **if** (self.exist\_edge(x, y)):  
 self.costs[(x, y)]=cost  
 **return True  
 return False  
  
def** add\_vertex(self,x):  
 **# adds a vertex to the graph, if this doesn't already exist. if it exists, it exits the function  
 # preconditions: x - an integer, a vertex**  
  
 **if** self.exist\_vertex(x):  
 **return False** self.ins[x]=[]  
 self.outs[x]=[]  
 **return True  
  
def** indegree(self,x):  
 **# parses the indegree of a vertex, if this already exists exists. if not, it exits the function  
 # preconditions: x - an integer, a vertex**  
 **if** self.exist\_vertex(x)==**False**:  
 **return False  
 return** len(self.ins[x])  
  
**def** outdegree(self,x):  
 **# parses the outdegree of a vertex, if this already exists exists. if not, it exits the function  
 # preconditions: x - an integer, a vertex**  
  
 **if** self.exist\_vertex(x)==**False**:  
 **return False  
 return** len(self.outs[x])  
  
**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** parse\_out(self,x):  
 **#parses the outbound edges of a vertex, and returns the list of the vertexes that go out, 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.outs[x]  
  
**def** remove\_edge(self,x,y):  
 **#removes an edge from the graph, if this already exists. if not, it exits the function  
 #preconditions: x,y - integers, representing 2 vertexes, x being the beginning of the edge, and y the final**  
 **if** self.exist\_edge(x,y)==**False**:  
 **return False** self.ins[y].remove(x)  
 self.outs[x].remove(y)  
 **del** self.costs[(x, y)]  
 **return True  
  
def** remove\_vertex(self,x):  
 **#removes a vertex from the graph, if this already exists. if not, it exits the function  
 #preconditions: x - an integer, a vertex to be removed**  
  
 **if** self.exist\_vertex(x)==**False**:  
 **return False  
 for** i **in** self.parse\_out(x):  
 self.remove\_edge(x,i)  
 **for** i **in** self.parse\_in(x):  
 self.remove\_edge(i,x)  
 **del** self.ins[x]  
 **del** self.outs[x]  
 **return True**