uc3m | Universidad Carlos III de Madrid

Data Structures and Algorithms.
Author: Isabel Segura Bedmar

Unit 6 – Graphs

Problem - Implement a graph using a Python dictionary.

Solution:

```
class Vertex:
    def __init__(self, vertex):
      """This constructor function takes the id of the vertes and creates
      and empty list to store its adjacent vertices"""
      self.id = vertex
      self.neighbors = []

    def addNeighbor(self, vertex):
      """This functions takes the id of a vertex
      adds its id as a new adjacent vertex for the vertex"""
      if vertex.id not in self.neighbors:
          #we only have to save the id of the vertex
          self.neighbors.append(vertex.id)
          self.neighbors = sorted(self.neighbors)


    def getNeighbors(self):
      """Returns the list of adjacent vertices"""
      return self.neighbors

    def __str__(self):
      return str(self.id) + ' connectedTo: ' + str([x for x in
self.neighbors])

class Graph:
    def __init__(self):
        self.vertices = {}

    def addVertex(self,key):
        newVertex = Vertex(key)
        self.vertices[key] = newVertex
```

```
    def getVertex(self,n):
        if n in self.vertices:
            return self.vertices[n]
        else:
            return None

    def getAdjacents(self,n):
        if n in self.vertices:
            return self.vertices[n].neighbors
        else:
            return None

    def __contains__(self,n):
        return n in self.vertices

    def addEdge(self,start,end):
        if start not in self.vertices:
            self.addVertex(start)
        if end not in self.vertices:
            self.addVertex(end)
        self.vertices[start].addNeighbor(self.vertices[end])

    def getVertices(self):
        return self.vertices.keys()

    def __iter__(self):
        return iter(self.vertices.values())

    def __str__(self):
        result=''
        for x in iter(self):
            result+=str(x)+'\n'

        return result



g = Graph()
for i in range(6):
  g.addVertex(i)

print(str(g))


g.addEdge(0,1)
g.addEdge(0,5)
g.addEdge(1,2)
g.addEdge(2,3)
g.addEdge(3,4)
g.addEdge(3,5)
```

```
g.addEdge(4,0)
g.addEdge(5,4)
g.addEdge(5,2)

print(str(g))
```

Problem: Implement the breadth-first and depth-first traversals. Please, check that your implementations are right for different graphs.

Solution:

```
    def bfs(self):
        visited=set()
        for v in self.vertices:
            if v not in visited:
                self._bfs(v,visited)

    def _bfs(self, start,visited):
        q = [start]
        while q:
            vertex = q.pop(0)
            if vertex not in visited:
                print(vertex)
                visited.add(vertex)
                adj=self.getAdjacents(vertex)
                for x in adj:
                    if x not in visited:
                        q.append(x)



   def dfs(self):
        visited=set()
        for v in self.vertices:
            if v not in visited:
                self._dfs(v,visited)

    def _dfs(self, vertex,visited):
        print(vertex)
        visited.add(vertex)
        adj=self.getAdjacents(vertex)
        for x in adj:
            if x not in visited:
                self._dfs(x,visited)
```