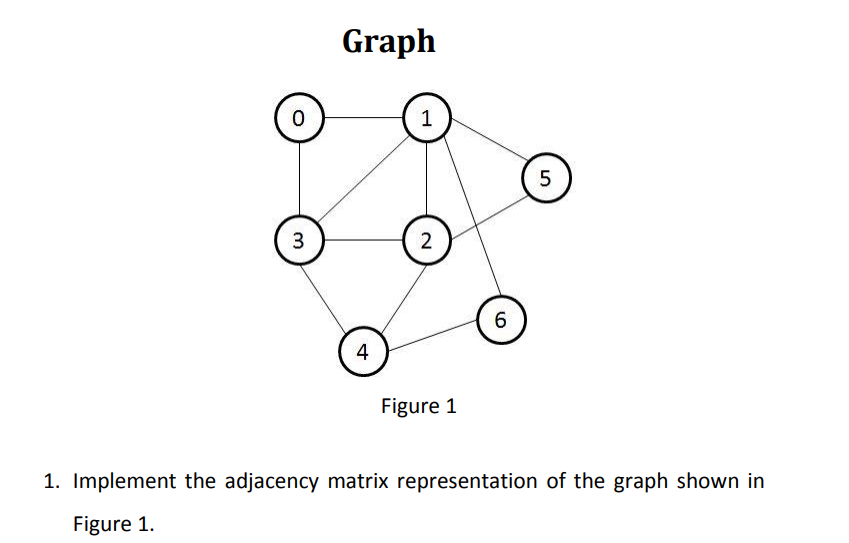
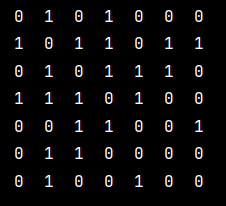
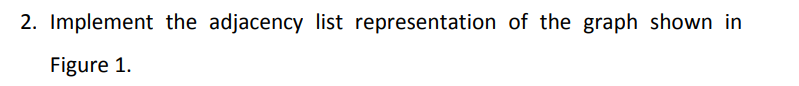
**S Abhishek AM.EN.U4CSE19147**

**Data Structures**

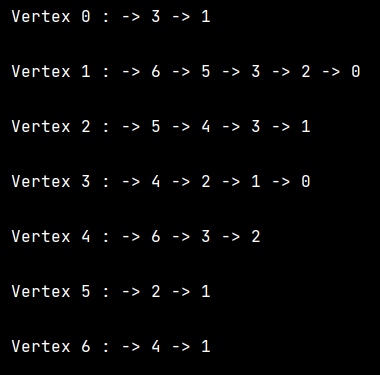
****

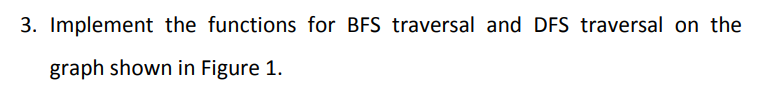
class Graph(object):  
  
 def \_\_init\_\_(self**,** row\_col):  
 self.arr = []  
 for i in range(row\_col):  
 self.arr.append([**0** for i in range(row\_col)])  
 self.row\_col = row\_col  
  
 def add(self**,** v1**,** v2):  
 if v1 == v2:  
 print("Same vertex %d and %d" % (v1**,** v2))  
 self.arr[v1][v2] = **1** self.arr[v2][v1] = **1** def display(self):  
 for row in self.arr:  
 for val in row:  
 print('{}'.format(val)**,**end=" ")  
 print()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 graph = Graph(**7**)  
 graph.add(**0, 1**)  
 graph.add(**0, 3**)  
 graph.add(**1, 2**)  
 graph.add(**1, 3**)  
 graph.add(**1, 5**)  
 graph.add(**1, 6**)  
 graph.add(**2, 3**)  
 graph.add(**2, 4**)  
 graph.add(**2, 5**)  
 graph.add(**3, 4**)  
 graph.add(**4, 6**)  
  
 graph.display()



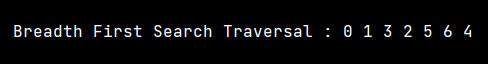


class Node:  
 def \_\_init\_\_(self**,** value):  
 self.vertex = value  
 self.next = None  
  
  
class Graph:  
 def \_\_init\_\_(self**,** data):  
 self.V = data  
 self.graph = [None] \* self.V  
  
 def add(self**,** x**,** y):  
 node = Node(y)  
 node.next = self.graph[x]  
 self.graph[x] = node  
  
 node = Node(x)  
 node.next = self.graph[y]  
 self.graph[y] = node  
  
 def display(self):  
 for i in range(self.V):  
 print("Vertex " + str(i) + " :"**,** end="")  
 ptr = self.graph[i]  
 while ptr:  
 print(" -> {}".format(ptr.vertex)**,** end="")  
 ptr = ptr.next  
 print(" \n")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 graph = Graph(**7**)  
 graph.add(**0, 1**)  
 graph.add(**0, 3**)  
 graph.add(**1, 2**)  
 graph.add(**1, 3**)  
 graph.add(**1, 5**)  
 graph.add(**1, 6**)  
 graph.add(**2, 3**)  
 graph.add(**2, 4**)  
 graph.add(**2, 5**)  
 graph.add(**3, 4**)  
 graph.add(**4, 6**)  
  
 graph.display()





from collections import defaultdict  
  
  
class Node:  
 def \_\_init\_\_(self):  
 self.graph = defaultdict(list)  
  
 def add(self**,** u: int**,** v: int):  
 self.graph[u].append(v)  
 self.graph[v].append(u)  
  
 def bfs(self**,** source: int):  
  
 bfs\_traverse = []  
  
 is\_visited = [False] \* len(self.graph)  
 queue = [source]  
  
 is\_visited[source] = True  
  
 while len(queue) > **0**:  
  
 curr\_node = queue.pop(**0**)  
 bfs\_traverse.append(curr\_node)  
  
 for neighbour\_node in self.graph[curr\_node]:  
 if not is\_visited[neighbour\_node]:  
 queue.append(neighbour\_node)  
 is\_visited[neighbour\_node] = True  
  
 return bfs\_traverse  
  
  
def run\_bfs(node: Node**,** source: int):  
 return node.bfs(source)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 graph = Node()  
 graph.add(**0, 1**)  
 graph.add(**0, 3**)  
 graph.add(**1, 2**)  
 graph.add(**1, 3**)  
 graph.add(**1, 5**)  
 graph.add(**1, 6**)  
 graph.add(**2, 3**)  
 graph.add(**2, 4**)  
 graph.add(**2, 5**)  
 graph.add(**3, 4**)  
 graph.add(**4, 6**)  
  
 bfs\_traverse = run\_bfs(graph**, 0**)  
  
 print("\nBreadth First Search Traversal : "**,** end="")  
 print(' '.join(str(ele) for ele in bfs\_traverse))



from collections import defaultdict  
  
  
class Node:  
 def \_\_init\_\_(self):  
  
 self.graph = defaultdict(list)  
  
 def add(self**,** u: int**,** v: int):  
 self.graph[u].append(v)  
 self.graph[v].append(u)  
  
 def dfs(self**,** source: int):  
  
 dfs\_traverse = []  
  
 is\_visited = [False] \* len(self.graph)  
 stack = [source]  
  
 is\_visited[source] = True  
  
 curr\_node = source  
 while len(stack) > **0**:  
 dfs\_traverse.append(curr\_node)  
  
 flag\_found\_next = False  
  
 while not flag\_found\_next and len(stack) > **0**:  
  
 for neighbour\_node in self.graph[curr\_node]:  
 if not is\_visited[neighbour\_node]:  
 # make visited True as they join queue  
 is\_visited[neighbour\_node] = True  
 stack.append(neighbour\_node)  
 curr\_node = neighbour\_node  
 flag\_found\_next = True  
 break  
  
 if not flag\_found\_next and len(stack):  
 curr\_node = stack.pop()  
  
 return dfs\_traverse  
  
  
def run\_dfs(node: Node**,** source: int):  
 return node.dfs(source)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 graph = Node()  
 graph.add(**0, 1**)  
 graph.add(**0, 3**)  
 graph.add(**1, 2**)  
 graph.add(**1, 3**)  
 graph.add(**1, 5**)  
 graph.add(**1, 6**)  
 graph.add(**2, 3**)  
 graph.add(**2, 4**)  
 graph.add(**2, 5**)  
 graph.add(**3, 4**)  
 graph.add(**4, 6**)  
  
 dfs\_traverse = run\_dfs(graph**, 0**)  
  
 print("Depth First Search Traversal : "**,** end="")  
 print(' '.join(str(ele) for ele in dfs\_traverse))



**One Drive :** [**Click Me!!**](https://amritauniv-my.sharepoint.com/:f:/g/personal/sabhishek_am_students_amrita_edu/Es4j-9Ia6ZJHnlwUccEeu08Bc9mUWXlmNkdLVHedARi0gg?e=LJAmkU)

**Thankyou!**