**AI/ML**

**5 BCA B**

**"Practical - 2"**

***BY***

**"Tharan" (23215134)**

**SUBMITTED TO**

**MEGHAVI**

****

**SCHOOL OF SCIENCES**

**2025-2026**

## **QUESTION 1**

**1. Traverse the tree using BFS**

### **Code Solution**

target = 6

### **FINAL Output**



## **QUESTION 2**

**2. Showcase the process of searching element in a graph.**

### **Code Solution**

class Node:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.left = None  
 self.right = None  
  
def bfs\_traversal(root):  
 if not root:  
 return []  
 result = []  
 queue = []  
 queue.append(root)  
 while queue:  
 current = queue.pop(0)  
 result.append(current.value)  
 if current.left:  
 queue.append(current.left)  
 if current.right:  
 queue.append(current.right)  
 return result  
  
root = Node(1)  
root.left = Node(2)  
root.right = Node(3)  
root.left.left = Node(4)  
root.left.right = Node(5)  
root.right.left = Node(6)  
root.right.right = Node(7)  
traversal\_result = bfs\_traversal(root)

### **FINAL Output**



## **QUESTION 3**

**3. Take input from the user for searching an element.**

### **Code Solution**

class Graph:  
 def \_\_init\_\_(self):  
 self.graph = {}  
   
 def add\_edge(self, u, v):  
 if u not in self.graph:  
 self.graph[u] = []  
 self.graph[u].append(v)  
  
 def dfs\_search(self, start, target, visited=None):  
 if visited is None:  
 visited = set()  
   
 visited.add(start)  
   
 if start == target:  
 return True  
   
 if start in self.graph:  
 for neighbor in self.graph[start]:  
 if neighbor not in visited:  
 if self.dfs\_search(neighbor, target, visited):  
 return True  
 return False  
  
 def bfs\_search(self, start, target):  
 visited = set()  
 queue = []  
   
 queue.append(start)  
 visited.add(start)  
   
 while queue:  
 vertex = queue.pop(0)  
   
 if vertex == target:  
 return True  
   
 if vertex in self.graph:  
 for neighbor in self.graph[vertex]:  
 if neighbor not in visited:  
 queue.append(neighbor)  
 visited.add(neighbor)  
 return False  
  
g = Graph()  
g.add\_edge(0, 1)  
g.add\_edge(0, 2)  
g.add\_edge(1, 2)  
g.add\_edge(2, 0)  
g.add\_edge(2, 3)  
g.add\_edge(3, 3)  
  
target\_element = 3  
print(g.dfs\_search(2, target\_element))  
print(g.bfs\_search(2, target\_element))

### **FINAL Output**

