**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  
take multiple malues to form a tree forst then traverse**

### **Code Solution**

search\_element = 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 root is None:  
 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**

def build\_graph():  
 graph = {  
 'A': ['B', 'C'],  
 'B': ['A', 'D', 'E'],  
 'C': ['A', 'F'],  
 'D': ['B'],  
 'E': ['B', 'F'],  
 'F': ['C', 'E']  
 }  
 return graph  
  
def bfs\_search(graph, start, target):  
 visited = set()  
 queue = [[start]]  
   
 if start == target:  
 return [start]  
   
 while queue:  
 path = queue.pop(0)  
 node = path[-1]  
   
 if node not in visited:  
 for neighbor in graph[node]:  
 new\_path = list(path)  
 new\_path.append(neighbor)  
 queue.append(new\_path)  
   
 if neighbor == target:  
 return new\_path  
 visited.add(node)  
 return []  
  
def dfs\_search(graph, start, target, path=None, visited=None):  
 if path is None:  
 path = []  
 if visited is None:  
 visited = set()  
   
 path.append(start)  
 visited.add(start)  
   
 if start == target:  
 return path  
   
 for neighbor in graph[start]:  
 if neighbor not in visited:  
 result = dfs\_search(graph, neighbor, target, path, visited)  
 if result:  
 return result  
   
 path.pop()  
 return []  
  
graph = build\_graph()  
target\_element = 'F'  
start\_node = 'A'  
  
bfs\_result = bfs\_search(graph, start\_node, target\_element)  
dfs\_result = dfs\_search(graph, start\_node, target\_element)  
  
print(f"BFS Path: {' -> '.join(bfs\_result)}")  
print(f"DFS Path: {' -> '.join(dfs\_result)}")

### **FINAL Output**

