**NAME        FATIMA AKBAR**

**ROLL NO      072**

**TASK NO      06**

**CLASS        BSDS(3A)**

**Lab 6 Tasks**

**BFS Traversals (Without Queue & With Queue):**

**Purpose of Code:**

The purpose of this lab is to understand and implement Breadth First Search (BFS) traversal of a binary tree in two different ways:  
1. BFS without using a Queue (using recursion and level-order traversal).  
2. BFS with Queue (using list as a queue).  
The tree is represented using a dictionary instead of a Node class.

**Concepts Used:**

- BFS (Breadth First Search): Visits nodes level by level starting from the root.  
- Dictionary Representation: Tree nodes are stored as dictionary keys with values as [LeftChild, RightChild].  
- Recursion: Used in BFS without Queue to print nodes level by level.  
- Queue (List in Python): Used in BFS with Queue to maintain order of node visits.

**Python Code:**  
tree = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F', 'G'],  
 'D': [None, None],  
 'E': [None, None],  
 'F': [None, None],  
 'G': [None, None]  
}  
  
def print\_level(tree, node, level):  
 if node is None:  
 return  
 if level == 1:  
 print(node, end=" ")  
 elif level > 1:  
 print\_level(tree, tree[node][0], level - 1)  
 print\_level(tree, tree[node][1], level - 1)  
  
def height(tree, node):  
 if node is None:  
 return 0  
 left\_h = height(tree, tree[node][0])  
 right\_h = height(tree, tree[node][1])  
 return max(left\_h, right\_h) + 1  
  
def bfs\_without\_queue(tree, root):  
 h = height(tree, root)  
 for i in range(1, h + 1):  
 print\_level(tree, root, i)  
  
print("BFS without Queue:")  
bfs\_without\_queue(tree, 'A')  
  
def bfs\_with\_queue(tree, root):  
 queue = [root] # start with root  
 while queue:  
 node = queue.pop(0) # dequeue  
 print(node, end=" ")  
 if tree[node][0]:  
 queue.append(tree[node][0])  
 if tree[node][1]:  
 queue.append(tree[node][1])  
  
print("\nBFS with Queue:")  
bfs\_with\_queue(tree, 'A').