**AI LAB – ASSIGNMENT**

**NAME: ABDULLAH**

**ROLL NO: Su92-BSAIM-F24-002**

**SECTION: BSAI-3A**

# Lab 6 Tasks - BFS

## Task 1: BFS without Queue & without using node:

In this task, we implement Breadth First Search (BFS) without using a Queue and without using a Node class. We represent the graph using an adjacency list (dictionary). Instead of using a queue, we simulate FIFO behavior using list slicing.

### Python Code:

def bfs(graph, start):  
 visited = set()  
 to\_visit = [start]  
  
 while len(to\_visit) > 0:  
 vertex = to\_visit[0]  
 to\_visit = to\_visit[1:]  
  
 if vertex not in visited:  
 print(vertex, end=" ")  
 visited.add(vertex)  
  
 for neighbor in graph[vertex]:  
 if neighbor not in visited:  
 to\_visit.append(neighbor)  
  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
print("BFS Traversal:")  
bfs(graph, 'A')

### Explanation:

1. Graph is stored as adjacency list.  
2. 'to\_visit' list stores nodes to be visited (works like a queue).  
3. We always take the first element using to\_visit[0] and then remove it using slicing.  
4. Mark the node visited and print it.  
5. Add all its unvisited neighbors to the list.  
6. Continue until no nodes remain.  
  
Output: A B C D E F

## Task 2: BFS with Queue:

In this task, we implement BFS using a Queue. We use a Python list as a queue. We use append() to add at the end and pop(0) to remove from the front (FIFO behavior).

### Python Code:

def bfs(graph, start):  
 visited = set()  
 queue = [start]  
  
 while queue:  
 vertex = queue.pop(0)  
  
 if vertex not in visited:  
 print(vertex, end=" ")  
 visited.add(vertex)  
  
 for neighbor in graph[vertex]:  
 if neighbor not in visited:  
 queue.append(neighbor)  
  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
print("BFS Traversal:")  
bfs(graph, 'A')

### Explanation:

1. Graph is represented as adjacency list.  
2. A list 'queue' is used to maintain nodes in FIFO order.  
3. The first element is removed using pop(0).  
4. If the node is not visited, print it and mark visited.  
5. Add all unvisited neighbors to the queue.  
6. Repeat until queue is empty.  
  
Output: A B C D E F