

**BS Artificial Intelligence**

**Name:** Hafiz M. Muneeb Akbar

**Roll No:** SU92-BSAIM-F24-048

**Subject:** Artificial Intelligence (Lab)

**Lab Task 6**

**Step-by-Step Explanation of Code (BFS without Queue and Node, BFS with Queue and Node)**

**This document provides a step-by-step explanation of two Python tasks:**

1. **Breadth-First Search (BFS) without using Queue and Node**, implemented through recursion.
2. **Breadth-First Search (BFS) with Queue and Node**, implemented using an explicit queue data structure.

**Task 1: BFS without Queue and Node**

Breadth-First Search (BFS) is a graph traversal technique. In this task, BFS is implemented **without using a queue**; instead, recursion is used to handle the traversal of each level.

**Function Definition**  
A function named **without\_Queue** is defined which accepts the graph (tree) and a starting node (source) as input.

**Explored List**  
An empty list named explored is created to keep track of already visited nodes.

**Recursive Function traverse**  
Inside **without\_Queue**, a helper function traverse is defined. It processes all nodes at the current level.

**Base Case Check**  
If the list of nodes at the current level (**level\_nodes**) is empty, the recursion stops.

**Processing Current Level**  
For each node in **level\_nodes**:

* If the node has not been explored, it is printed and added to the explored list.
* All its adjacent nodes are collected into the list **nxt** for processing in the next level.

**Recursive Call**  
The function traverse**(nxt)** is called recursively to continue BFS with the next level of nodes.

**Starting the Traversal**  
The traversal begins by calling traverse**([source]),** passing the source node inside a list.

**Example Execution**  
When called as **without\_Queue(net, 'A'),** the algorithm prints nodes of the graph in BFS order without using a queue structure.

**Output Screenshot**



**Task 2: BFS with Queue and Node**

This task performs Breadth-First Search (BFS) using an explicit queue data structure to ensure nodes are visited level by level.

**Graph Input**  
The graph is represented as a dictionary where each key is a node and its value is the list of adjacent nodes.

**Function Definition**  
A function named **with\_Queue** is defined which accepts the **graph (net)** and a source **node (src)** as input.

**Visited List**  
An empty list named seen is initialized to keep track of all visited nodes.

**Queue Initialization**  
A list q is created with the source node as its first element, representing the BFS queue.

**While Loop**  
The algorithm continues as long as the queue is not empty.

**Process Current Node**

* The first element of the queue is removed using **pop(0**) and stored in **curr.**
* If **curr** is not already visited, it is printed and added to seen.

**Add Neighbors**  
For each neighbor of **curr** in the graph:

* If the neighbor has not been visited and is not already in the queue, it is appended to the queue.

**Output**  
The nodes are printed in BFS order using the queue to ensure **FIFO (First In, First Out)** traversal.

**Output Screenshot**

