#### **Name: Javeria Manzoor**

**Roll no: BSAI-3C-201**

**Submitted to: Sir Rasikh**

**Lab task:06**

**Subject:Artificial Intelligence**

#### **Code Explanation – BFS Without and With Queue Introduction:**

#### **This program demonstrates two different implementations of the Breadth-First Search (BFS) algorithm in Python. Both methods produce the same output but use different techniques:**

#### **Recursive BFS (without a queue)**

#### **Iterative BFS (using a queue)**

#### **BFS is used to traverse a graph level by level, starting from a given node and visiting all its directly connected neighbors before moving to the next level.**

### **1. BFS Without Queue:**

#### In this version, BFS is implemented recursively using a function that explores nodes at a specific level. It does not use a queue, but recursion achieves the same level-order traversal.

#### **How the Code Works:** The graph is represented using a Python dictionary, where each key is a node and its value is a list of its neighboring nodes. Example: 'A': ['B', 'C']

#### means node A is connected to nodes B and C.

#### Function: nodes\_at\_level(graph, start, level)

#### This function returns a list of nodes that are exactly level edges away from the start node.

#### If the level is 0, it means we are at the starting node, so it simply returns [start].

#### Otherwise, it recursively calls itself for each neighbor while decreasing the level by 1.

#### Finally, it combines the results from all recursive calls to build a list of nodes at that level.

#### **Why this works:** By reducing the level value in each recursive call, the function effectively “steps down” through the graph structure, ensuring that only nodes at the desired depth are returned. Function: bfs\_without\_queue(graph, start) Starts with level = 0 and an empty visited set to track visited nodes.

#### In each iteration: It calls nodes\_at\_level() to get nodes at the current level. Filters out nodes already visited.

#### If no new nodes are found, the loop ends. Otherwise, it prints the nodes, adds them to visited, and increments the level by 1. The loop continues until all levels of the graph have been processed.

### **2. BFS With Queue**

#### This is the standard iterative implementation of BFS, which uses a queue to manage the order of node visits.

#### **How the Code Works:**

#### The same dictionary structure is used — each node maps to its adjacent nodes. 1:Function: bfs\_with\_queue(graph, start) Imports deque from collections because it provides fast queue operations (append and popleft).

#### Initialize:

#### A visited set (to avoid revisiting nodes)

#### A queue (q) containing the start node.

#### While the queue is not empty:

#### Removes the front node (popleft()).

#### If the node is unvisited:

#### Prints the node and marks it as visited.

#### Adds all its unvisited neighbors to the queue.

#### This continues until the queue is empty, meaning all reachable nodes have been visited.

#### **Why this works:** The queue ensures nodes are visited in the exact order they were discovered, which is the essence of BFS. Each node is processed once, and its neighbors are explored before moving deeper into the graph.

#### 

#### 