## **Name: Javeria Manzoor**

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

**Subject:Artificial Intelligence(LAB)**

**Task no:3**

**Submitted to : Sir Rasikh**

## **QUESTION 1: BFS Without Queue**

### **How it Works:**

**Graph Representation** The graph is stored as a **dictionary** where each node has a list of its neighbors:  
  
 graph = {'A': ['B', 'C'], 'B': ['D', 'E'], ...}

**nodes\_at\_level()**

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

if level == 0:

return [start]

1. When level = 0, it returns the starting node itself.
2. Otherwise, it recursively calls itself for all neighbors, reducing the level by 1 each time.
3. This collects **all nodes present at a particular level** (distance from the start).  
   **bfs\_without\_queue()**

**Function:**

Starts from level 0 and keeps increasing the level number.

At each level:

It finds all nodes at that level (nodes\_at\_level)

Removes already visited nodes.

Prints new nodes and marks them as visited.  
Stops when no more unvisited nodes remain.

### **Why it Works:**

* Instead of a **queue**, this version uses **recursion** to explore the graph level by level.
* Each recursive call finds nodes that are exactly level steps away from the start node.
* It still follows the **BFS principle** (visit all neighbors level-wise), just implemented differently.



## **QUESTION 2: BFS With Queue**

### **How it Works:**

1. **Graph Representation** Same dictionary structure as before.
2. **bfs\_with\_queue() Function** q = deque([start])

visited = set()

1. Uses a **queue (deque)** to store nodes that need to be visited.  
   Starts by adding the first node (start) to the queue.

**Main Loop** while q:

node = q.popleft()

1. Takes out the **front element** from the queue.
2. If the node hasn’t been visited:
3. Prints it.
4. Marks it as visited.
5. Adds its **unvisited neighbors** to the queue.

**Why it Works:**

* The **queue** ensures nodes are visited in **FIFO (First In, First Out)** order — just like BFS should.
* This means:
* You visit a node,
* Then visit all its neighbors,
* Then move to the next node in order.

