**Name: Sundas Shoukat**

**Roll no.057**

**Section: BSDS-3A**

**Explanation of BFS without Queue**

**How the Code Works (Step by Step)**

1. **Graph Representation**
   * The graph is stored in a dictionary.
   * Each node (key) has a list of its connected nodes (values).
2. **Visited List**
   * A visited list keeps track of all the nodes that have already been visited.
   * This ensures that nodes are not repeated.
3. **Level List Instead of Queue**
   * Instead of using a queue, a list called level is used.
   * level contains the nodes of the current level in the graph.
   * At the start, it only contains the starting node.
4. **While Loop**
   * The loop runs until the level list becomes empty.
   * For each node in the current level, check if it is visited:
     + If not, print it and mark it as visited.
     + Collect all its children and put them in a new list called next\_level.
5. **Moving to Next Level**
   * After finishing one level, level is replaced by next\_level.
   * This means the algorithm now processes the next set of nodes.
6. **Process Continue**
   * The process repeats until there are no nodes left.
   * The graph is traversed level by level, just like BFS.

**Why This Approach is Used**

* **Why Without Queue:**  
  Normally, BFS uses a queue, but here the same behavior is achieved by using lists (level and next\_level).  
  This avoids the need for an external queue structure.
* **Why Visited List:**  
  Prevents visiting the same node multiple times, ensuring correct traversal.
* **Why BFS Works This Way:**
  + Using level and next\_level, the algorithm still explores the graph **level by level**.
  + It shows how BFS can be implemented even without explicitly using a queue.

**Output:**

****

**Explanation of BFS with Queue**

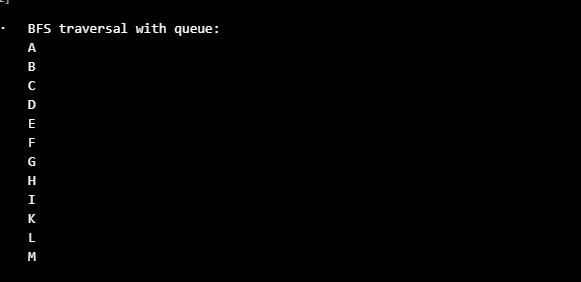
**How the Code Works (Step by Step)**

1. **Graph Representation**
   * The graph is represented as a dictionary.
   * Each key is a node, and the value is a list of its connected nodes (children).
2. **Visited List**
   * visited is a list used to store nodes that have already been visited.
   * This prevents revisiting the same node multiple times.
3. **Queue Initialization**
   * A queue is created using deque([start]), with the start node already inside it.
   * The queue is used because BFS follows the **First In, First Out (FIFO)** rule.
4. **While Loop**
   * The loop continues until the queue becomes empty.
   * In each iteration, one node is removed from the queue using popleft().
5. **Visit and Print Node**
   * If the current node has not been visited:
     + Print (or process) the node.
     + Add it to the visited list.
6. **Add Children to Queue**
   * Using the graph dictionary, the children of the current node are found.
   * Each child is added to the queue, so they will be processed later.
7. **Process Continue**
   * This continues until all nodes are visited and the queue is empty.
   * The result is a **level-by-level traversal** of the graph.

**Why This Approach is Used**

* **Why Queue is Used:**  
  The queue ensures that nodes are visited in the correct order (FIFO), which is the foundation of BFS.
* **Why Visited List is Used:**  
  To make sure each node is visited only once. Without this, the algorithm could go into an infinite loop.
* **Why BFS is Useful:**
  + BFS explores nodes **level by level**.
  + It is very useful in finding the **shortest path** in an unweighted graph.
  + It is reliable for tree and graph traversal problems.

**Output:**

****