**Abdull Haseeb**

**SU92-BSAIM-F24-050**

**BSAI-3A**

**bfs\_without\_queue (Breadth First Search without Explicit Queue)**

**Function Name:**

**bfs\_without\_queue(graph, start)**

**Purpose:**

**This function performs Breadth First Search (BFS) traversal of a graph without using a queue data structure.  
Instead, it simulates queue behavior using a list (bfs\_list) and an index pointer.**

**Parameters:**

1. **graph (dict):**
   * **Graph stored as an adjacency list.**
   * **Each node (key) has a list (value) of its neighbors.**

**Example:**

**{**

**'A': ['B', 'C'],**

**'B': ['D', 'E'],**

**'C': ['F'],**

**'D': [],**

**'E': ['F'],**

**'F': []**

**}**

1. **start (any type, usually string):**
   * **The starting node for BFS traversal.**

**Return Value:**

* **list:  
  A list of nodes visited in BFS order.**

**Working / Steps:**

1. **Initialization:**
   * **A visited set keeps track of visited nodes.**
   * **A bfs\_list is initialized with the starting node.**
   * **An index variable simulates queue front behavior.**
2. **Traversal Loop:**
   * **While index < len(bfs\_list):**
     + **Take the current node from bfs\_list using the index.**
     + **Add it to the result list.**
     + **Check its neighbors:**
       - **If a neighbor is not visited, mark it visited and append it to bfs\_list.**
     + **Increment the index to move to the next node.**
3. **Queue Simulation:**
   * **Instead of pop(0) (which is inefficient), this method increases an index pointer to mimic queue behavior.**

**Example Execution:**

**Graph:**

**A → [B, C]**

**B → [D, E]**

**C → [F]**

**D → []**

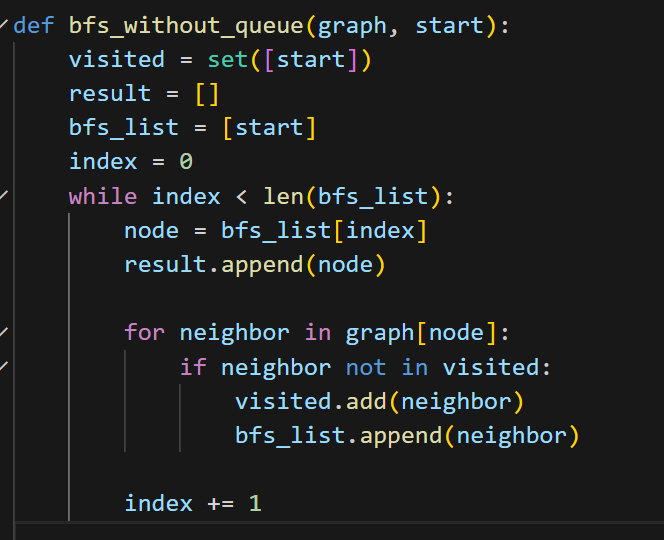
**E → [F]**

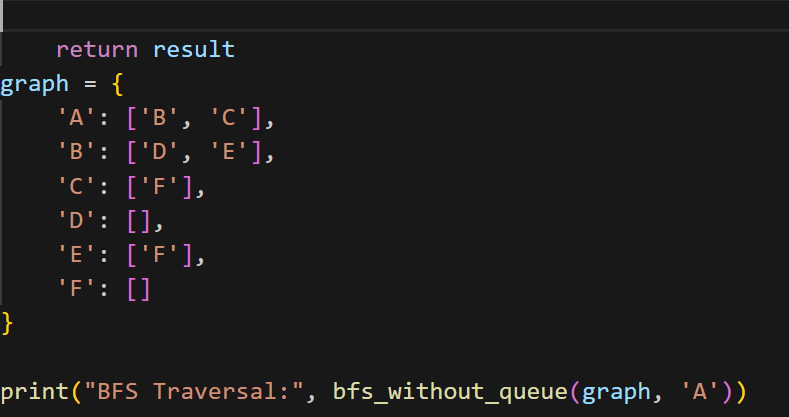
**F → []**

**Start = 'A'**

**BFS Traversal Output:**

**BFS Traversal: ['A', 'B', 'C', 'D', 'E', 'F']**

****

****

**bfs (Breadth First Search with Queue)**

**Function Name:**

**bfs(graph, start)**

**Purpose:**

**This function implements the Breadth First Search (BFS) algorithm using a queue.  
It traverses the graph level by level, starting from the given node, and prints the nodes in BFS order.**

**Parameters:**

1. **graph (dict):**
   * **The graph is represented as an adjacency list.**
   * **Each key is a node, and its value is a list of neighboring nodes.**

**Example:**

**{**

**'A': ['B', 'C'],**

**'B': ['D', 'E'],**

**'C': ['F'],**

**'D': [],**

**'E': ['F'],**

**'F': []**

**}**

1. **start (any type, usually string):**
   * **The node from which BFS traversal begins.**

**Working / Steps:**

1. **Visited Set:**
   * **A set is used to keep track of visited nodes to avoid revisiting them.**
2. **Queue Initialization:**
   * **A queue is created with the starting node inside it.**
   * **BFS uses a queue because it follows the FIFO (First In, First Out) principle.**
3. **Traversal Loop:**
   * **While the queue is not empty:**
     + **Remove (popleft) the front node.**
     + **If the node has not been visited:**
       - **Print the node (visit it).**
       - **Mark it as visited.**
       - **Add all its unvisited neighbors to the queue.**

**Example Execution:**

**Graph:**

**A → [B, C]**

**B → [D, E]**

**C → [F]**

**D → []**

**E → [F]**

**F → []**

**Start = 'A'**

**BFS Traversal Output:**

**A B C D E F**

****