

## BFS

```
from collections import deque
```

```
# Undirected graph as an adjacency list
```

```
graph = {
```

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

```
    'B': ['A', 'D', 'E'],
```

```
    'C': ['A', 'F'],
```

```
    'D': ['B'],
```

```
    'E': ['B', 'F'],
```

```
    'F': ['C', 'E']
```

```
}
```

```
visited = set()
```

```
def bfs_recursive(queue):
```

```
    if not queue:
```

```
        return
```

```
    node = queue.popleft()
```

```
    if node not in visited:
```

```
        print(node, end=' ')
```

```
        visited.add(node)
```

```
        for neighbor in graph[node]:
```

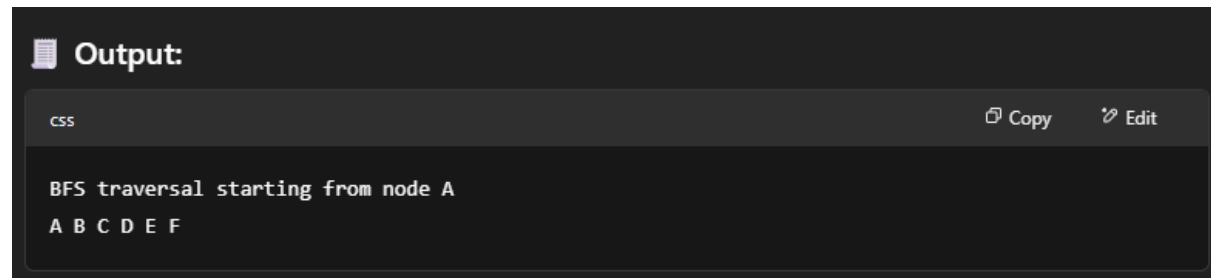
```
            if neighbor not in visited:
```

```
                queue.append(neighbor)
```

```
    bfs_recursive(queue)
```

### # Initialize queue and start BFS

```
start_node = 'A'  
  
print("BFS traversal starting from node", start_node)  
  
bfs_recursive(deque([start_node]))
```

A screenshot of a code editor with a dark theme. At the top, there's a tab labeled 'css'. Below the tab, there's a header 'Output:' with a document icon. The main area shows the output of a BFS traversal: 'BFS traversal starting from node A' followed by 'A B C D E F' on the next line. There are 'Copy' and 'Edit' buttons in the top right corner of the output area.

```
Output:  
  
BFS traversal starting from node A  
A B C D E F
```

### Viva Questions for This Practical

- 1. What is BFS (Breadth First Search) and how does it work?**
  - Explain the basic concept of BFS and how it explores all nodes at the current level before moving on to nodes at the next level.
- 2. What data structure is used to represent the graph in this code?**
  - The graph is represented using an adjacency list, implemented as a dictionary where each node is mapped to a list of its neighbors.
- 3. What is the role of the queue in BFS?**
  - The queue is used to manage the nodes to be processed. BFS explores nodes in a level-by-level manner, so the queue ensures that nodes are processed in the correct order.
- 4. What is the importance of the visited set in this implementation?**
  - The visited set prevents revisiting the same node, ensuring that the algorithm does not enter an infinite loop or process the same node multiple times.
- 5. How does the recursive implementation of BFS work?**
  - The recursive BFS function processes the first node in the queue, marks it as visited, adds its unvisited neighbors to the queue, and then recursively calls itself until the queue is empty.
- 6. What will happen if the graph has cycles?**
  - If there are cycles, the algorithm will still work correctly because the visited set ensures that each node is processed only once, preventing infinite loops.

**7. Can BFS be implemented iteratively?**

- Yes, BFS is typically implemented iteratively using a queue. The recursive version here simulates the iterative process with function calls, but the iterative implementation uses a while loop to process nodes.

**8. What is the time complexity of BFS?**

- The time complexity of BFS is  $O(V + E)$ , where  $V$  is the number of vertices and  $E$  is the number of edges in the graph.

**9. What is the space complexity of this BFS implementation?**

- The space complexity is  $O(V)$  due to the queue and the visited set.

**10. What happens if you start the BFS from a different node, say node 'F'?**

- The traversal will start from 'F' and will print the nodes in a different order depending on the graph's structure, but it will still visit all reachable nodes.

**11. What are the differences between BFS and DFS (Depth First Search)?**

- BFS explores nodes level by level (breadth-first), while DFS explores as deeply as possible along a branch before backtracking. BFS uses a queue, while DFS uses a stack (or recursion).

**12. How would you modify this BFS implementation to store the traversal path in a list instead of printing it?**

- Instead of printing node, you could append the node to a list (e.g., `path.append(node)`).

**13. How would you handle a weighted graph with BFS?**

- BFS can be applied to weighted graphs, but it does not account for weights. To consider weights, you might need to modify the algorithm or use a different algorithm like Dijkstra's.

**14. Can BFS be used to find the shortest path in an unweighted graph?**

- Yes, BFS can be used to find the shortest path in an unweighted graph because it explores all nodes at a given distance before moving on to nodes further away.

**15. What will happen if the graph is disconnected?**

- If the graph is disconnected, BFS will only visit the nodes connected to the starting node. To visit all nodes in a disconnected graph, we would need to run BFS from each unvisited node.

**16. What happens if the queue is empty at the start of the algorithm?**

- If the queue is empty at the start, the BFS algorithm will not process any nodes and terminate immediately.

**17. What is the role of the deque data structure in BFS?**

- The deque is used for efficient popping from the front and appending to the back, which is important for maintaining the correct order of node processing in BFS.

**18. How would you handle a situation where the graph contains self-loops?**

- Self-loops (edges that connect a node to itself) would not cause any issues in this BFS implementation because the visited set ensures that the node is not processed again, even if there is a self-loop.