# Breadth-First Search (BFS) - With and Without Queue

## 1. BFS with Queue

Breadth-First Search (BFS) is a graph traversal algorithm that explores all the vertices of a graph level by level.   
It starts from a selected source node and visits all its neighbors before moving to the next level.

### Code Implementation:

from collections import deque  
  
def bfs(graph, start):  
 visited = set()  
 queue = deque([start])  
  
 while queue:  
 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)  
  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
print("BFS Traversal:")  
bfs(graph, 'A')

### Explanation:

1. BFS uses a queue (FIFO structure) to process nodes level by level.  
2. The algorithm dequeues a node, visits it, and enqueues all its unvisited neighbors.  
3. This continues until all reachable nodes are visited.  
4. The order of traversal for the given graph is: A B C D E F

### Output:

BFS Traversal:  
A B C D E F

## 2. BFS without Using Queue

In this approach, we perform BFS without explicitly using a queue.   
Instead, we use two lists – one for the current level and another for the next level.   
This simulates the queue behavior while maintaining level order traversal.

### Code Implementation:

def bfs\_without\_queue(graph, start):  
 visited = []  
 level = [start]  
  
 while level:  
 next\_level = []  
 for node in level:  
 if node not in visited:  
 print(node, end=" ")  
 visited.append(node)  
 for neighbor in graph[node]:  
 if neighbor not in visited:  
 next\_level.append(neighbor)  
 level = next\_level  
  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
print("BFS Traversal Without Using Queue:")  
bfs\_without\_queue(graph, 'A')

### Explanation:

1. The list 'level' stores all nodes at the current depth level.  
2. For every node in 'level', we visit it and collect all unvisited neighbors in 'next\_level'.  
3. After processing one level, we move to the next by assigning level = next\_level.  
4. This continues until all levels are processed.  
5. The traversal order remains the same: A B C D E F

### Output:

BFS Traversal Without Using Queue:  
A B C D E F