# Lab Report: Breadth First Search (BFS)

## Introduction

Breadth First Search (BFS) is a fundamental graph traversal algorithm. It explores nodes level by level, starting from the root node and visiting all its neighbors before moving to the next level. BFS can be implemented in multiple ways, such as using a simple list or a queue data structure.

## Purpose of the Code

The purpose of this code is to demonstrate two implementations of BFS:  
1. BFS without Queue (using a list).  
2. BFS with Queue (using the deque data structure from collections).  
  
Both approaches show how BFS explores nodes systematically, ensuring that each node is visited only once.

## BFS without Queue and Node

1. def bfs\_without\_queue(graph, start):

- Defines the BFS function without explicitly using a queue. Instead, a simple list (to\_visit) is used to keep track of nodes to visit.

2. visited = []

- A list that stores nodes which have already been visited.

3. to\_visit = [start]

- Initialize the list with the starting node.

4. while to\_visit:

- Loop continues until there are no more nodes left to visit.

5. ver = to\_visit[0] and to\_visit = to\_visit[1:]

- The first element of the list is taken as the current node and removed from the list.

6. if ver not in visited:

- Ensures that nodes are not visited more than once.

7. print(ver, end=' ')

- Prints the current node as part of the traversal order.

8. visited.append(ver)

- Marks the current node as visited.

9. to\_visit.extend(graph[ver])

- Adds all the neighbors of the current node to the list for future exploration.

## BFS with Queue and Node

1. from collections import deque

- Importing deque, an efficient queue structure from Python's collections module.

2. def bfs\_with\_queue(graph, start):

- Defines the BFS function that explicitly uses a queue for node management.

3. visited = []

- List to store visited nodes.

4. queue = deque([start])

- Initialize queue with the starting node.

5. while queue:

- Loop continues until the queue is empty.

6. ver = queue.popleft()

- Dequeue the first element (FIFO behavior ensures BFS order).

7. if ver not in visited:

- Checks if the node has already been visited.

8. print(ver, end=' ') and visited.append(ver)

- Prints the node and marks it as visited.

9. queue.extend(graph[ver])

- Adds neighbors of the current node into the queue for exploration.

## Graph Structure

The graph is represented as a dictionary. Each key is a node, and its value is a list of neighboring nodes.

graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': ['G','H'],  
 'E': [],  
 'F': ['I','K'],  
 'G': [],  
 'H': ['L'],  
 'I': [],  
 'K': ['M'],  
 'L': [],  
 'M': []  
}

## Output

For both implementations, starting from node 'A', the traversal will explore level by level. The expected output is:  
  
BFS Traversal (Without Queue and Node): A B C D E F G H I K L M  
BFS Traversal (With Queue & Node): A B C D E F G H I K L M

## Conclusion

In this lab, we implemented BFS in two different ways. The first approach used a simple list to mimic queue behavior, while the second approach used the deque structure for efficient queue operations. Both methods successfully demonstrated BFS traversal and helped us understand the difference between queue-based and list-based implementations.