Breadth-First Search (BFS) Traversal

This lab demonstrates the Breadth-First Search (BFS) algorithm using two different approaches:

1. **BFS with Queue (Iterative Implementation)**
2. **BFS without Queue (Recursive Levels Implementation)**

1. BFS with Queue (Iterative)

In this implementation, a queue data structure is used to keep track of nodes.
Nodes are dequeued one by one, added to the path, and their children are enqueued.
This ensures that nodes are explored **level by level**.

Code:

```
class GraphTraversal:

    def __init__(self, graph):
        self.graph = graph
        self.path = []

    def bfs_with_queue(self, start, goal):
        queue = [start]
        self.path = []
        print(f"Starting BFS with Queue from: {start}, and the Goal is :
{goal}")

        while queue:
            current = queue.pop(0)
            print(f"Dequeued: {current}, Current Path: {self.path}")
            self.path.append(current)

            if current == goal:
                print(f"Goal {goal} found. Final Path: {self.path}")
                return self.path

            print(f"Expanding {current}, adding children:
{self.graph[current]}")
            queue.extend(self.graph[current])
            print(f"Queue now: {queue}")

        print(f"Goal {goal} not found.")
        print('Final output: ', self.path)
        return False
```

Explanation:

- Starts with the root node in the queue.

- Iteratively dequeues nodes and explores their children.
- Ensures breadth-first traversal by using a queue.
- Stops when the goal is found or when all nodes are explored.

2. BFS without Queue (Recursive Levels)

In this implementation, recursion is used instead of a queue.
Nodes are explored level by level by passing the **current level** of nodes as a parameter, and generating the next level recursively.

Code:

```
class GraphTraversal:
    def __init__(self, graph):
        self.graph = graph
        self.path = []

    def bfs_no_queue(self, levels, goal):
        print(f"Exploring Level: {levels}, Current Path: {self.path}")
        if not levels:
            print(f"No more levels to explore. Goal {goal} not found.")
            return False

        next_level = []
        for node in levels:
            print(f"Visiting: {node}")
            self.path.append(node)

            if node == goal:
                print(f"Goal {goal} found. Final Path: {self.path}")
                return self.path

            print(f"Adding children of {node}: {self.graph[node]}")
            next_level.extend(self.graph[node])

        return self.bfs_no_queue(next_level, goal)
```

Explanation:

- Starts by visiting nodes of the current level.
- For each node, its children are collected into the next level.
- Recursively explores the next level until the goal is found or no levels remain.

Difference Between Both Implementations

1. BFS with Queue uses an **iterative approach** and an explicit queue data structure.
2. BFS without Queue uses a **recursive approach** where recursion maintains levels.
3. Both ensure breadth-first traversal but differ in the way they handle levels:
    - Queue explicitly stores nodes to be explored.
    - Recursion implicitly handles levels without an explicit queue.
4. Queue-based BFS is more common and efficient for large graphs.
5. Recursive BFS is elegant but may face recursion depth limits in very large graphs.