**Documentation: Breadth-First Search (BFS) Implementations**

**Introduction**

Breadth-First Search (BFS) is a fundamental graph traversal algorithm used in computer science to explore nodes layer by layer. It starts at a source node, visits all of its immediate neighbors, then moves on to their neighbors, and so on. BFS guarantees the shortest path in an unweighted graph and is commonly used in pathfinding, web crawling, and network analysis.

This document explains two different implementations of BFS:

1. **Recursive BFS using adjacency lists**
2. **BFS using an object-oriented Node class with a queue**

**1. Recursive BFS with Adjacency List**

**Structure**

* The graph is represented as an **adjacency list** using a Python dictionary.
* Each key is a node, and its value is a list of connected nodes (children).

**Process**

* **Input Parameters**: The function takes three inputs:
  + adj\_list: the graph structure
  + current\_level: the list of nodes to process at the current depth
  + seen: a collection of visited nodes
* **Traversal Logic**:
  + If there are no more nodes at the current level, recursion ends.
  + Each node in the current level is checked:
    - If not visited, it is processed (printed or stored).
    - Its children are added to the next level’s frontier.
  + The function recursively calls itself with the next level of nodes.

**Advantages**

* Simple and elegant in small graphs.
* Highlights the layer-by-layer exploration of BFS.

**Limitations**

* Uses recursion, which is not memory-efficient for large graphs.
* May cause a recursion depth error for deep or complex graphs.

**2. BFS with Queue and Node Class**

**Structure**

* A custom Node class is defined with two attributes:
  + value: the identifier or data of the node
  + children: a list of connected child nodes
* A queue (deque from Python’s collections) manages the traversal order.
* A seen list keeps track of already visited nodes.

**Process**

1. The root node is inserted into the queue.
2. While the queue is not empty:
   * The first node is removed from the queue.
   * If it has not been visited, it is processed.
   * All of its children are added to the queue.
3. This continues until all nodes are visited.

**Advantages**

* More intuitive and widely used BFS structure.
* Queue-based approach is memory-efficient and avoids recursion limits.
* Fits naturally with tree or graph data structures using objects.

**Limitations**

* Requires explicit handling of visited nodes.
* Slightly more verbose due to the object-oriented setup.

**Comparison of Both Approaches**

| **Feature** | **Recursive BFS (Adjacency List)** | **BFS with Queue (Node Class)** |
| --- | --- | --- |
| **Data Structure** | Dictionary (Adjacency List) | Class-based tree/graph nodes |
| **Traversal Mechanism** | Recursion | Iteration with queue |
| **Memory Usage** | Higher (recursion overhead) | Lower, efficient with queue |
| **Ease of Understanding** | Concise but less common | Standard and widely taught |
| **Scalability** | Limited (stack depth) | Scales well to large graphs |

**Applications of BFS**

* **Shortest Path**: Finding the shortest path in unweighted graphs (e.g., social networks, maps).
* **Tree Traversal**: Exploring hierarchical data such as organizational charts.
* **Web Crawling**: Layered crawling of pages starting from a root URL.
* **Network Broadcasting**: Simulating message spreading in computer networks.