Breadth-First Search (BFS) Algorithm

Breadth-First Search (BFS) is a fundamental graph traversal algorithm that explores a graph level by level. It starts at a selected node (the "source" or "start node") and explores all of its immediate neighbors. Then, for each of those neighbors, it explores their unexplored neighbors, and so on, until all reachable nodes have been visited.
Because of its level-by-level approach, BFS is particularly useful for finding the shortest path between two nodes in an unweighted graph.

How the Algorithm Works

The BFS algorithm relies on a **queue** data structure, which follows a First-In, First-Out (FIFO) principle. This ensures that nodes are processed in the order they are discovered.
The process can be broken down into the following steps:

1. **Initialization:**
    - Create a **queue** and add the start_node to it.
    - Create a **visited set** to keep track of nodes that have already been visited. Add the start_node to this set to prevent it from being visited again.
    - Create a **visited_order list** to store the sequence of visited nodes.
2. **Traversal Loop:**
    - The algorithm loops as long as the **queue** is not empty.
    - In each iteration, **dequeue** the node at the front of the queue. This becomes the current_node.
    - Add the current_node to the visited_order list.
3. **Neighbour Exploration:**
    - For each **neighbor** of the current_node:
    - Check if the neighbor has been visited by looking it up in the visited set.
    - If the neighbor has **not** been visited:
        - Mark it as visited by adding it to the visited set.
        - **Enqueue** the neighbor, adding it to the back of the queue for future processing.
4. **Completion:**
    - Once the queue is empty, it means all reachable nodes from the start_node have been visited.
    - The algorithm returns the visited_order list, which contains the nodes in the order of their traversal.

Example

Input Graph:

```
graph = {
   'A': ['B', 'C'],
   'B': ['A', 'D', 'E'],
   'C': ['A', 'F'],
   'D': ['B'],
   'E': ['B', 'F'],
   'F': ['C', 'E','G'],
   'G':['H'],
   'H':['G']
}
```

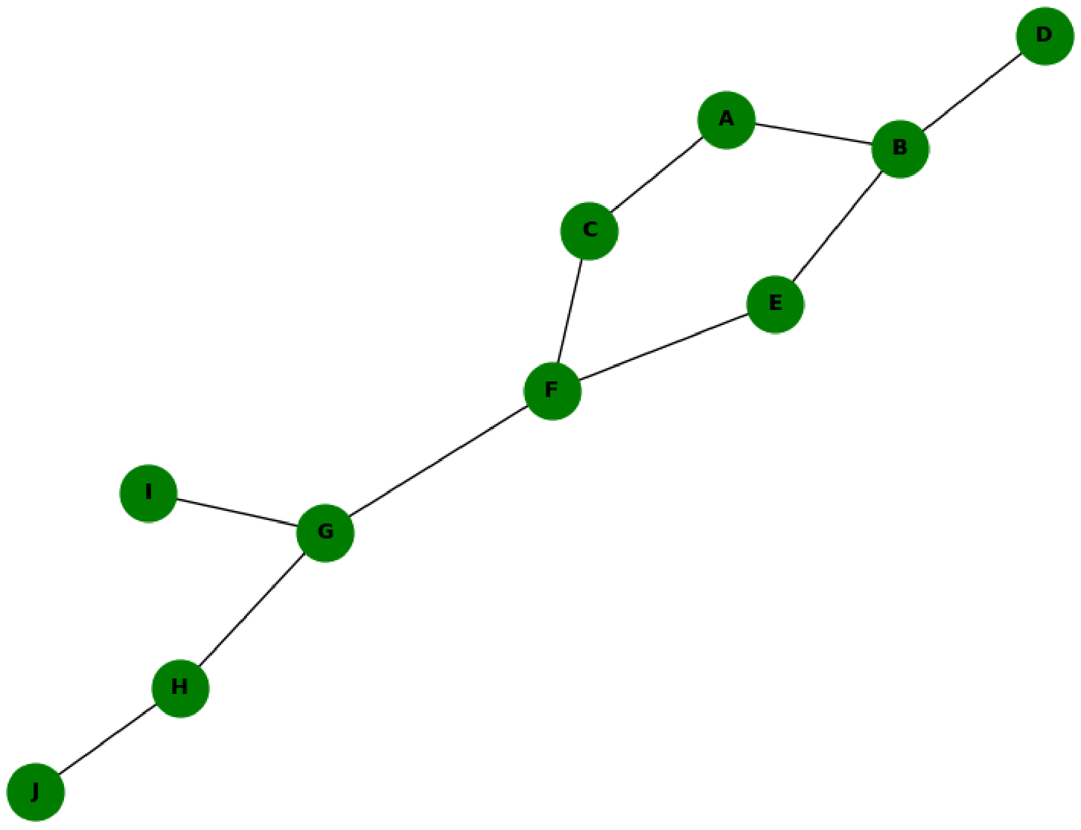
Traversal starting from node 'A':

1. **Initialize:** queue = ['A'], visited = {'A'}, visited_order = []
2. **Dequeue 'A'**: visited_order = ['A']. Neighbors of 'A' are 'B' and 'C'.
3. **Enqueue 'B' and 'C'**: queue = ['B', 'C'], visited = {'A', 'B', 'C'}.
4. **Dequeue 'B'**: visited_order = ['A', 'B']. Neighbors of 'B' are 'A', 'D', 'E'. 'A' is visited.
5. **Enqueue 'D' and 'E'**: queue = ['C', 'D', 'E'], visited = {'A', 'B', 'C', 'D', 'E'}.
6. **Dequeue 'C'**: visited_order = ['A', 'B', 'C']. Neighbors of 'C' are 'A', 'F'. 'A' is visited.
7. **Enqueue 'F'**: queue = ['D', 'E', 'F'], visited = {'A', 'B', 'C', 'D', 'E', 'F'}.
8. **Dequeue 'D'**: visited_order = ['A', 'B', 'C', 'D']. No unvisited neighbors.
9. **Dequeue 'E'**: visited_order = ['A', 'B', 'C', 'D', 'E']. Neighbors are 'B', 'F'. Both are visited.
10. **Dequeue 'F'**: visited_order = ['A', 'B', 'C', 'D', 'E', 'F']. Neighbor 'G' is not visited.
11. **Enqueue 'G'**: queue = ['G'], visited = {'A', 'B', 'C', 'D', 'E', 'F', 'G'}.
12. **Dequeue 'G'**: visited_order = ['A', 'B', 'C', 'D', 'E', 'F', 'G']. Neighbor 'H' is not visited.
13. **Enqueue 'H'**: queue = ['H'], visited = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'}.
14. **Dequeue 'H'**: visited_order = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']. Neighbor 'G' is visited.
15. The queue is now empty. The traversal is complete.

Final Output:
Graph representation (Adjacency List):
{'A': ['B', 'C'], 'B': ['A', 'D', 'E'], 'C': ['A', 'F'], 'D': ['B'], 'E': ['B', 'F'], 'F': ['C', 'E', 'G'], 'G': ['H'], 'H': ['G']}

BFS traversal starting from node 'A':
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

Graph Visualization



Complexity Analysis

- **Time Complexity: O(V + E)**
  This is because every vertex (V) and every edge (E) will be explored exactly once. The algorithm dequeues each vertex once and checks all its edges.
- **Space Complexity: O(V)**
  The space required is determined by the maximum number of nodes stored in the queue and the visited set at any given time. In the worst-case scenario (a star graph, for example), the queue may hold up to V-1 nodes, leading to a space complexity proportional to the number of vertices.