### Algorithm: Breadth-First Search (BFS) for Shortest Path in Unweighted Graph

BFS can be used to find the shortest path in an unweighted graph, where the shortest path is the path with the fewest edges.

#### Initialize Data Structures:

* Use a queue to keep track of nodes to visit.
* Use a set to keep track of visited nodes.
* Use a dictionary to keep track of the distance to each node.

#### Traverse:

* Enqueue the root node and set its distance to 0.
* Dequeue a node, process it, and enqueue its unvisited neighbors with updated distances.

#### Retrieve the Result:

* The distance dictionary contains the shortest distances from the start node to each node.

#### Implementation:

| **from** **collections** **import** deque  **def** bfs\_shortest\_path(graph: Dict[int, List[int]], start: int) -> Dict[int, int]:  visited = set()  queue = deque([start])  distances = {start: 0}  **while** queue:  node = queue.popleft()  **if** node **not** **in** visited:  visited.add(node)  **for** neighbor **in** graph[node]:  **if** neighbor **not** **in** visited:  queue.append(neighbor)  distances[neighbor] = distances[node] + 1  **return** distances  *# Example usage:*  graph = {  1: [2, 3],  2: [4, 5],  3: [6, 7],  4: [],  5: [],  6: [],  7: []  }  start = 1  print(bfs\_shortest\_path(graph, start)) *# Output: Shortest path distances* |
| --- |

#### Explanation:

Initialize:

* queue: A queue initialized with the start node.
* distances: A dictionary initialized with the start node and distance 0.

Traverse:

* Dequeue a node, process it, and enqueue its unvisited neighbors with updated distances.

Retrieve the Result:

* The distance dictionary contains the shortest distances from the start node to each node.