

PROMPT 1 ----->

```
record Graph (Map<Integer, List<Integer>> adjacentList, int numberOfNodes) {

    public Graph (int numberOfNodes) {
        this(new HashMap<>(), numberOfNodes);
        for (int node = 0; node < numberOfNodes; node++)
            this.adjacentList.put(node, new ArrayList<>());
    }

    public void addEdge (int node1, int node2) {
        this.adjacentList.get(node1).add(node2);
        this.adjacentList.get(node2).add(node1);
    }

    public List<Integer> traverseBFS (int sourceNode) {
        List<Integer> result = new ArrayList<>();

        boolean[] visited = new boolean[this.numberOfNodes]; // Keeps track of node visited as parent node.
        Queue<Integer> nodeQueue = new LinkedList<>();
        nodeQueue.add(sourceNode);

        while (!nodeQueue.isEmpty()) {
            int currentNodeCount = nodeQueue.size();

            while (currentNodeCount-- > 0) {
                int node = nodeQueue.remove();
                visited[node] = true;
                result.add(node);

                List<Integer> neighbourNodeList = this.adjacentList.get(node);
                for (int neighbourNode : neighbourNodeList) {
                    if (visited[neighbourNode]) continue;
                    nodeQueue.add(neighbourNode);
                }
            }
        }

        return result;
    }
}

// Is the Breadth First Search function "int traverseBFS (int sourceNode)" for graph traversal is correct or
// not, Can you please confirm?
```

DeepSeek AI Response ----->

The BFS (Breadth-First Search) implementation in your `traverseBFS` method has a **logical error** that can lead to **incorrect traversal and duplicate nodes** in the result. Here's the issue:

### ### Problem:

When you add a node's neighbors to the queue, you **don't mark them as visited immediately**, which means the same neighbor node can be added to the queue multiple times by different parent nodes before it gets processed. This can happen if multiple nodes share the same unvisited neighbor.

### ### Example:

Consider this simple undirected graph:

```

\ \
0 -- 1
|   |
3 -- 2
\ \

```

- If you start BFS from `0`, the queue will initially have `[0]`.
- When processing `0`, its neighbors `1` and `3` are added to the queue (`[1, 3]`), but they are not yet marked as visited.
- When processing `1`, its neighbor `2` is added (`[3, 2]`), and when processing `3`, its neighbor `2` is added again (`[2, 2]`), resulting in duplicates.

### Fix:  
You should **mark a node as visited as soon as it is added to the queue**, not when it is processed. This ensures that no node is enqueued more than once.

### Corrected Code:

```
```java
public List<Integer> traverseBFS(int sourceNode) {
    List<Integer> result = new ArrayList<>();
    boolean[] visited = new boolean[this.numberOfNodes];
    Queue<Integer> nodeQueue = new LinkedList<>();

    // Mark the source node as visited and add it to the queue
    visited[sourceNode] = true;
    nodeQueue.add(sourceNode);

    while (!nodeQueue.isEmpty()) {
        int currentNodeCount = nodeQueue.size();

        while (currentNodeCount-- > 0) {
            int node = nodeQueue.remove();
            result.add(node); // Add to result when dequeued

            // Process neighbors
            for (int neighbourNode : this.adjacentList.get(node)) {
                if (!visited[neighbourNode]) {
                    visited[neighbourNode] = true; // Mark as visited immediately
                    nodeQueue.add(neighbourNode);
                }
            }
        }
    }
    return result;
}
```
```

### Key Changes:

1. **Mark `sourceNode` as visited before the loop**.
2. **Mark neighbors as visited when they are enqueued**, not when they are processed.
3. This ensures no duplicates and correct BFS order.

Now your BFS implementation is correct! 🚀

PROMPT 2 ----->

```
record Graph (Map<Integer, List<Integer>> adjacentList, int numberOfNodes) {

    public Graph (int numberOfNodes) {
        this(new HashMap<>(), numberOfNodes);
        for (int node = 0; node < numberOfNodes; node++)
            this.adjacentList.put(node, new ArrayList<>());
    }

    public void addEdge (int node1, int node2) {
        this.adjacentList.get(node1).add(node2);
        this.adjacentList.get(node2).add(node1);
    }

    public List<Integer> traverseBFS (int sourceNode) {
        List<Integer> result = new ArrayList<>();

        boolean[] visited = new boolean[this.numberOfNodes]; // Keeps track of nodes visited.
        Queue<Integer> nodeQueue = new LinkedList<>();
        nodeQueue.add(sourceNode);
        visited[sourceNode] = true; // Mark here [1].

        while (!nodeQueue.isEmpty()) {
            int currentNodeCount = nodeQueue.size();

            while (currentNodeCount-- > 0) {
                int node = nodeQueue.remove();
```

```

        //visited[node] = true; // Error: Never mark here because same neighbour node may be visited by other
parent nodes also.
        result.add(node);

        List<Integer> neighbourNodeList = this.adjacentList.get(node);
        for (int neighbourNode : neighbourNodeList) {
            if (visited[neighbourNode]) continue;
            nodeQueue.add(neighbourNode);
            visited[neighbourNode] = true; // Mark here [2].
        }
    }

    }

    return result;
}
}

```

// Here is the code and I did the recommended correction. Is it correct now?

DeepSeek AI Response ----->

Yes! Your corrected BFS implementation is now **correct** and avoids duplicate nodes in the traversal. Here's why:

#### ### Key Improvements:

1. **Marking `sourceNode` as visited when enqueued** (`[1]`):  
- Prevents it from being added again if it reappears as a neighbor.
2. **Marking neighbors as visited when they are enqueued** (`[2]`):  
- Ensures no duplicate entries in the queue, even if multiple nodes share the same neighbor.

#### ### Why This Works:

- **BFS Property**: Nodes are processed level-by-level, and each node should be visited **only once**.
- **Early Visited Marking**: By marking a node as visited **as soon as it's discovered** (when added to the queue), you guarantee no future duplicates.

#### ### Example Walkthrough:

For the graph:

```

0 -- 1
|   |
3 -- 2

```

- **Step 1**: Start at `0` (marked visited, queue = `[0]`).
- **Step 2**: Dequeue `0`, add neighbors `1` and `3` (both marked visited immediately, queue = `[1, 3]`).
- **Step 3**: Dequeue `1`, add neighbor `2` (marked visited, queue = `[3, 2]`).
- **Step 4**: Dequeue `3`, skip neighbor `2` (already visited), queue = `[2]`.
- **Step 5**: Dequeue `2`, no unvisited neighbors left.

**Result**: `[0, 1, 3, 2]` (correct BFS order, no duplicates).

#### ### Final Notes:

- The commented line `// visited[node] = true; // Error...` is indeed unnecessary (and harmful) because it would delay marking and allow duplicates.
- Your fix aligns with standard BFS implementations (e.g., CLRS or competitive programming references).

### ✅ **Your code is now correct!** ✅