Problem Statement -1

Given a **undirected graph** represented by an adjacency list `adj`, where each `adj[i]` represents the list of vertices connected to vertex `i`. Perform a **Breadth First Search (BFS)** traversal starting from vertex `0`, visiting vertices from left to right according to the adjacency list, and return a list containing the BFS traversal of the graph.

Examples:***Input:*** *adj[][] = [[1,2], [0,2,3], [0,1,4], [1,4], [2,3]]****Output:*** *[0, 1, 2, 3, 4]*

Explanation: *Starting from 0, the BFS traversal will follow these steps:*

```
Visit 0 → Output: [0]
Visit 1 (first neighbor of 0) → Output: [0, 1]
Visit 2 (next neighbor of 0) → Output: [0, 1, 2]
Visit 3 (next neighbor of 1) → Output: [0, 1, 2, 3]
Visit 4 (neighbor of 2) → Final Output: [0, 1, 2, 3, 4]
```

Program:

```
import java.util.*;

public class Main {

    public static List<Integer> bfsOfGraph(int V, List<List<Integer>> adj) {

        List<Integer> bfs = new ArrayList<>();

        boolean[] visited = new boolean[V];

        Queue<Integer> queue = new LinkedList<>();

        queue.add(0);

        visited[0] = true;

        while (!queue.isEmpty()) {

            int node = queue.poll();

            bfs.add(node);
            for (int neighbor : adj.get(node)) {
                if (!visited[neighbor]) {
                    queue.add(neighbor);
                    visited[neighbor] = true;
                }
            }
        }

        return bfs;
    }
    public static void main(String[] args) {
        int V = 5;
        List<List<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < V; i++) {
            adj.add(new ArrayList<>());
        }
        adj.get(0).add(1);
        adj.get(0).add(2);
        adj.get(1).add(0);
        adj.get(1).add(2);
        adj.get(1).add(3);
        adj.get(2).add(0);
        adj.get(2).add(1);
        adj.get(2).add(4);
        adj.get(3).add(1);
        adj.get(3).add(4);
        adj.get(4).add(2);
        adj.get(4).add(3);
        List<Integer> result = bfsOfGraph(V, adj);
        System.out.println("BFS Traversal: " + result);
    }
}
```

Problem Statement -1

In Depth First Search (or DFS) for a graph, we traverse all adjacent vertices one by one. When we traverse an adjacent vertex, we completely finish the traversal of all vertices reachable through that adjacent vertex. This is similar to a tree, where we first completely traverse the left subtree and then move to the right subtree. The key difference is that, unlike trees, graphs may contain cycles (a node may be visited more than once). To avoid processing a node multiple times, we use a boolean visited array.

Example:

Note : *There can be multiple DFS traversals of a graph according to the order in which we pick adjacent vertices. Here we pick vertices as per the insertion order.*

Input: *adj = [[1, 2], [0, 2], [0, 1, 3, 4], [2], [2]]*

Output:* [0 1 2 3 4*]*
Explanation: *The source vertex s is 0. We visit it first, then we visit an adjacent.*
Start at 0: Mark as visited. Output: 0
Move to 1: Mark as visited. Output: 1
Move to 2: Mark as visited. Output: 2
Move to 3: Mark as visited. Output: 3 (backtrack to 2)
Move to 4: Mark as visited. Output: 4 (backtrack to 2, then backtrack to 1, then to 0)

Not that there can be more than one DFS Traversals of a Graph. For example, after 1, we may pick adjacent 2 instead of 0 and get a different DFS. Here we pick in the insertion order.

***PROGRAM*:**

```
import java.util.*;

public class Main {
    public static void dfs(int node, boolean[] visited, List<List<Integer>> adj,
List<Integer> result) {
        visited[node] = true;
        result.add(node);

        for (int neighbor : adj.get(node)) {
            if (!visited[neighbor]) {
                dfs(neighbor, visited, adj, result);
            }
        }
    }

    public static List<Integer> dfsOfGraph(int V, List<List<Integer>> adj) {
        boolean[] visited = new boolean[V];
        List<Integer> result = new ArrayList<>();
        dfs(0, visited, adj, result);
        return result;
    }

    public static void main(String[] args) {
        int V = 5;
        List<List<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < V; i++) {
            adj.add(new ArrayList<>());
        }
        adj.get(0).add(1);
        adj.get(0).add(2);
        adj.get(1).add(0);
        adj.get(1).add(2);
        adj.get(2).add(0);

        adj.get(2).add(1);

        adj.get(2).add(3);

        adj.get(2).add(4);

        adj.get(3).add(2);

        adj.get(4).add(2);

        List<Integer> result = dfsOfGraph(V, adj);

        System.out.print("DFS Traversal: ");

        for (int node : result) {

            System.out.print(node + " ");

        }

    }

}
```