


```

}

return false;
}

private boolean hasCycleBFS(int src, ArrayList<ArrayList<Integer>> adj, boolean[] vis) {
    Queue<int[]> q = new LinkedList<>(); // [node, parent]
    q.add(new int[]{src, -1});
    vis[src] = true;

    while (!q.isEmpty()) {
        int[] curr = q.poll();
        int node = curr[0], parent = curr[1];

        for (int neighbor : adj.get(node)) {
            if (!vis[neighbor]) {
                vis[neighbor] = true;
                q.add(new int[]{neighbor, node});
            } else if (neighbor != parent) {
                return true; // Visited + not parent = CYCLE
            }
        }
    }

    return false;
}
}

```

■ ■ Time & Space Complexity

Case	Time	Space	Why
Time	$O(V + E)$	$O(V)$	Each vertex and edge processed once
Space	$O(V)$	$O(V)$	Queue and visited array both $O(V)$

■ ■ Common Mistakes

Avoid These

- Not tracking parent — without parent, the reverse edge falsely triggers cycle detection.
- Using visited alone — in undirected, ALL edges look like they revisit the other end.
- For DFS version: pass parent as parameter, not a global variable.

■ Practice Problems

Case	Time	Space	Why
Medium	GFG	Detect Cycle Undirected	Direct application
Medium	LC 684	Redundant Connection	Find the cycle-creating edge
Medium	LC 261	Graph Valid Tree	Cycle + connectivity check