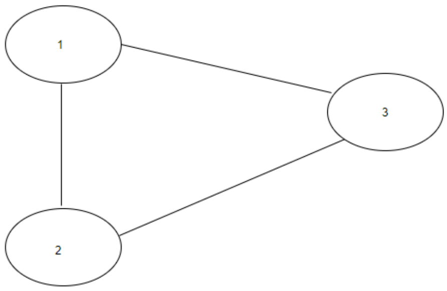
Cycle Detection in Undirected Graph CodeStudio

You have been given an undirected graph with 'N' vertices and 'M' edges. The vertices are labelled from 1 to 'N'.

Your task is to find if the graph contains a cycle or not.

Example:



Output: True

Explanation:

addEdge Function:

- **Purpose:**
  - Populates the graph's adjacency list based on the provided edge list.
- **Explanation:**
  - Iterates through each edge in the **edges** vector.
  - For each edge, extracts the source vertex **u** and iterates over the connected vertices.
  - Adds an edge from **u** to **v** and from **v** to **u** in the adjacency list.
- **Time Complexity:**
  - **O(E), where E is the number of edges in the input vector.**
- **Space Complexity:**
  - **O(E), where E is the number of edges. Each edge results in the creation of two entries in the adjacency list.**

Approach 1: Function to perform cycle detection using Breadth-First Search (BFS)

- **Purpose:**

    - Detects cycles in an undirected graph using BFS.
- **Explanation:**
    - Uses BFS to traverse the graph.
    - Maintains a visited map to keep track of visited nodes and a parent map for each node during BFS.
    - If, during BFS, a visited node is encountered that is not the parent of the current node, a cycle is detected.
- **Time Complexity:**
    - **O(V + E), where V is the number of vertices and E is the number of edges.**
    - **Combined with addEdge Function:**
        - **Total Time Complexity: O(V + E)**
- **Space Complexity:**
    - **O(V), where V is the number of vertices. Space is needed for the visited and parent maps.**
    - **Combined with addEdge Function:**
        - **Total Space Complexity: O(V + E)**

Approach 2: Function to perform cycle detection using Depth-First Search (DFS)

- **Purpose:**
    - Detects cycles in an undirected graph using DFS.
- **Explanation:**
    - Uses DFS to traverse the graph.
    - Calls a recursive helper function for each unvisited node.
    - If a cycle is detected in any DFS traversal, returns true.
- **Time Complexity:**
    - **O(V + E), where V is the number of vertices and E is the number of edges in the connected component.**
    - **Combined with addEdge Function:**
        - **Total Time Complexity: O(V + E)**
- **Space Complexity:**

- **O(V), where V is the number of vertices. Space is needed for the visited map.**
- **Combined with addEdge Function:**
    - **Total Space Complexity: O(V + E)**

Conclusion:

- Both approaches effectively detect cycles in undirected graphs.
- The **addEdge** function is a prerequisite for both approaches, and its complexities are combined with each approach's complexities.
- **The choice between BFS and DFS may depend on factors such as memory constraints, specific use cases, or preferences in terms of implementation.**