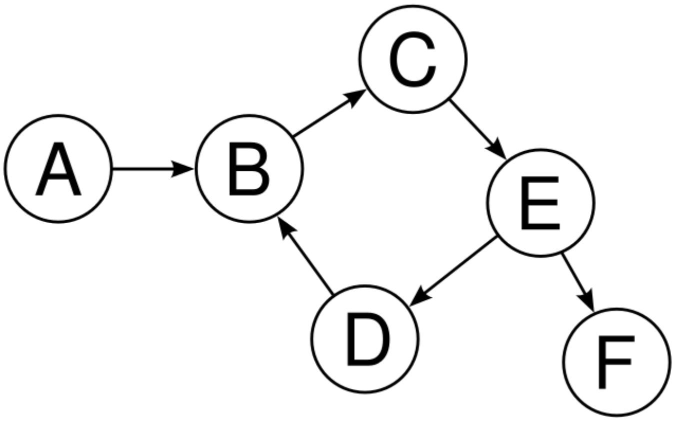
Cycle Detection in Directed Graph CodeStudio

You are given a directed graph having 'N' nodes. A matrix 'EDGES' of size M x 2 is given which represents the 'M' edges such that there is an edge directed from node EDGES[i][0] to node EDGES[i][1].

Find whether the graph contains a cycle or not, return true if a cycle is present in the given directed graph else return false.

Example:



Output: True

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** 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 an entry in the adjacency list.**

Approach 1: Function to detect cycles in a directed graph using BFS (Modified Kahn's Algorithm)

- **Purpose:**
    - Detects cycles in directed graphs using BFS.
- **Explanation:**
    - Utilizes in-degrees to identify nodes with no incoming edges and enqueues them.
    - Decreases in-degrees of neighbors during BFS traversal.
    - A directed graph has a cycle if and only if it is not a Directed Acyclic Graph (DAG).
- **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) + O(E) = O(V + 2E) ≈ O(V + E)**
- **Space Complexity:**
    - **O(V + E), where V is the number of vertices and E is the number of edges.**
    - **Combined with addEdge Function:**
        - **Total Space Complexity: O(V + E)**

Approach 2: Function to detect cycles in a directed graph using DFS

- **Purpose:**
    - Detects cycles in directed graphs using DFS.
- **Explanation:**
    - Employs a recursive DFS approach with two sets of visited flags (**visited** and **dfsVisited**).
    - A cycle is detected if a node is visited in the current DFS traversal.
- **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) + O(E) = O(V + 2E) ≈ O(V + E)**
- **Space Complexity:**

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

Conclusion:

- Both BFS and DFS approaches effectively detect cycles in directed graphs.
- The **addEdge** function is essential for establishing graph connections, contributing to the overall time and space complexity.
- The choice between BFS and DFS depends on specific requirements, with both approaches offering comparable performance.