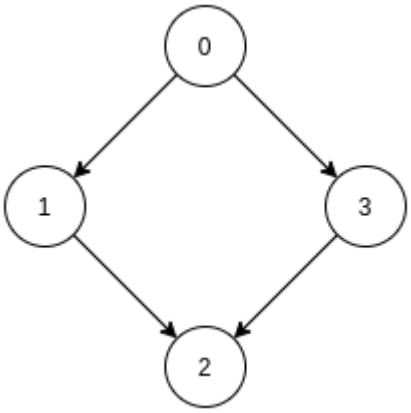
Topological Sort on Directed Acyclic Graph CodeStudio

A Directed Acyclic Graph (DAG) is a directed graph that contains no cycles.

Topological Sorting of DAG is a linear ordering of vertices such that for every directed edge from vertex ‘u’ to vertex ‘v’, vertex ‘u’ comes before ‘v’ in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

Given a DAG consisting of ‘V’ vertices and ‘E’ edges, you need to find out any topological sorting of this DAG. Return an array of size ‘V’ representing the topological sort of the vertices of the given DAG.

Example:



Output: {0, 1, 3, 2}

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 perform topological sort using depth-first search (DFS)

- **Purpose:**
  - Generates the topological ordering of nodes in a DAG using Depth-First Search (DFS).
- **Explanation:**
  - Utilizes a stack to store nodes in reverse order of their finishing times in DFS.
  - Performs DFS for each unvisited node in the graph.
- **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 perform topological sort using BFS (Kahn's Algorithm)

- **Purpose:**
  - Generates the topological ordering of nodes in a DAG using Breadth-First Search (BFS) and Kahn's Algorithm.
- **Explanation:**
  - Calculates in-degrees for each node and initializes a queue with nodes having in-degree zero.
  - Updates in-degrees during BFS traversal and enqueues nodes with in-degree zero.
- **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)**

Conclusion:

- Both DFS and BFS-based approaches yield correct topological orderings for a DAG.
- The choice between the two approaches depends on factors such as simplicity, implementation preference, or specific requirements.
- The **addEdge** function is a fundamental component contributing to the overall time and space complexity.