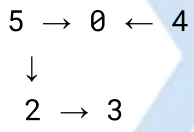
Concept of Topological Order

Topological Sorting of a **Directed Acyclic Graph (DAG)** is a **linear ordering of vertices** such that for every directed edge u → v, vertex u comes **before** v in the ordering.

Conditions:

- Only applicable to **Directed Acyclic Graphs (DAGs).**
- There can be **more than one valid topological order.**

Example Graph:



Valid Topological Orders:

- 4 5 2 3 0
- 5 2 3 4 0
- etc.

In all orders, **dependencies** are satisfied (for example, 5 comes before 0).

Kahn's Algorithm (BFS Based Topological Sort)

Idea:

- Count **in-degrees** of all nodes.
- Start with nodes having `in-degree = 0`.
- Remove them and reduce in-degrees of neighbors.
- Repeat until all nodes are processed.

C++ Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void kahnTopologicalSort(int V, vector<int> adj[]) {
    vector<int> inDegree(V, 0);

    // Step 1: Compute in-degree of each node
    for (int i = 0; i < V; i++) {
        for (int neighbor : adj[i])
            inDegree[neighbor]++;
    }

    queue<int> q;

    // Step 2: Enqueue all vertices with in-degree 0
    for (int i = 0; i < V; i++)
        if (inDegree[i] == 0)
            q.push(i);
```

```
    vector<int> topoOrder;

    // Step 3: Process the queue
    while (!q.empty()) {
        int curr = q.front(); q.pop();
        topoOrder.push_back(curr);

        for (int neighbor : adj[curr]) {
            inDegree[neighbor]--;
            if (inDegree[neighbor] == 0)
                q.push(neighbor);
        }
    }

    if (topoOrder.size() != V) {
        cout << "Cycle detected! Topological sort not possible.\n";
    } else {
        cout << "Topological Order (Kahn's): ";
        for (int node : topoOrder)
            cout << node << " ";
        cout << endl;
    }
}
```

DFS-Based Topological Sort

Idea:

- Visit nodes recursively.
- On returning from recursion, push node to a stack (postorder).
- Reverse the stack for topological order.

C++ Code:

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

void dfs(int node, vector<int> adj[], vector<bool>& visited,
stack<int>& st) {
    visited[node] = true;

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

    st.push(node); // Post-order push
}

void dfsTopologicalSort(int V, vector<int> adj[]) {
    vector<bool> visited(V, false);
    stack<int> st;

    for (int i = 0; i < V; i++)
```

```
        if (!visited[i])
            dfs(i, adj, visited, st);

    cout << "Topological Order (DFS): ";
    while (!st.empty()) {
        cout << st.top() << " ";
        st.pop();
    }
    cout << endl;
}
```

Applications of Topological Sorting

1. Task Scheduling

Example: Task A must be completed before B

- Topological order gives the valid sequence of execution.

2. Compilation Order

- Some files depend on others.
- Topological sort determines the correct order to compile.

3. Dependency Resolution

- Used in build systems like `Make`, `npm`, or `Maven`.

4. Course Prerequisites

- Courses that depend on other courses (DAG of course structure).