🔄 Cycle Detection in Directed Graphs

Using Kahn's Topological Sort Algorithm

🧠 How It Works

Key Principle: A directed graph has NO cycles if and only if it has a valid topological ordering.

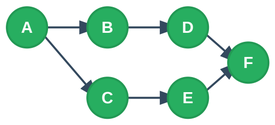
Kahn's Algorithm:

```
1. Calculate in-degree for each vertex (number of incoming edges) 2. Add all vertices with in-degree 0 to a queue 3. While queue is not empty: - Remove vertex from queue and add to topological order - For each neighbor of removed vertex: - Decrease its in-degree by 1 - If in-degree becomes 0, add to queue 4. If topological order contains ALL vertices → NO CYCLE If some vertices remain unprocessed → CYCLE EXISTS
```

Why this detects cycles: In a cyclic graph, vertices in the cycle will always have in-degree ≥ 1, so they can never be added to the queue and processed.

🔵 Unprocessed (in-degree > 0) 🟠 In Queue (in-degree = 0) 🟢 Processed

📈 Acyclic Graph (DAG)



Initial in-degrees: A(0), B(1), C(1), D(1), E(1), F(2)

Step 1: Process A → Queue: [B, C]

Step 2: Process B → Queue: [C, D]

Step 3: Process C → Queue: [D, E]

Step 4: Process D → Queue: [E, F]

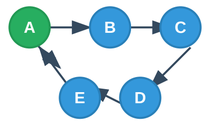
Step 5: Process E → Queue: [F]

Step 6: Process F → Queue: []

All 6 vertices processed successfully!

✅ NO CYCLE DETECTED
Topological Order: A → B → C → D → E → F

🔄 Cyclic Graph



Initial in-degrees: A(0), B(1), C(1), D(1), E(1)

Step 1: Process A → Queue: [B]

Step 2: Process B → Queue: [C]

Step 3: Process C → Queue: [D]

Step 4: Process D → Queue: [E]

Step 5: Process E → Queue: []

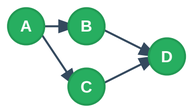
Wait! E points back to A, creating a cycle!

Only 5 vertices processed, but graph has 5 vertices

❌ CYCLE DETECTED
Cycle: B → C → D → E → (back to B)

🔍 Detailed Step-by-Step Analysis

Step-by-Step: Acyclic Detection



🎯 Initial: in-degree[A]=0, [B]=1, [C]=1, [D]=2

📥 Queue starts with: [A]

✅ Process A: Queue becomes [B, C]

✅ Process B: Queue becomes [C, D]

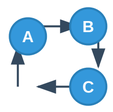
✅ Process C: Queue becomes [D]

✅ Process D: Queue becomes []

🎉 Processed 4/4 vertices → NO CYCLE!

✅ ACYCLIC GRAPH
Valid topological order exists

Step-by-Step: Cycle Detection



🎯 Initial: in-degree[A]=1, [B]=1, [C]=1

📥 Queue starts with: [] (no vertex has in-degree 0!)

⚠️ Queue is empty but vertices remain

🚫 Cannot process any vertex

🔄 All vertices are part of cycle A→B→C→A

💥 Processed 0/3 vertices → CYCLE FOUND!

❌ CYCLIC GRAPH
Cycle detected: A→B→C→A

⚡ Algorithm Analysis

Time Complexity: O(V + E) where V = vertices, E = edges

Space Complexity: O(V) for the queue and in-degree array

Applications:

- 🎓 Course prerequisite checking
- 🏗️ Build dependency resolution
- 📋 Task scheduling
- 🧬 Deadlock detection

💻 Implementation Pseudocode

```
function hasCycle(graph): in_degree = calculateInDegrees(graph) queue = [] processed_count = 0 // Add all vertices with in-degree 0 to queue for each vertex v: if in_degree[v] == 0: queue.add(v) while queue is not empty: current = queue.remove() processed_count++ for each neighbor of current: in_degree[neighbor]-- if in_degree[neighbor] == 0: queue.add(neighbor) // If we processed all vertices, no cycle exists return processed_count != total_vertices
```