| **NAME:** Hamza Butt | **SAP:** 55041 |
|---|---|

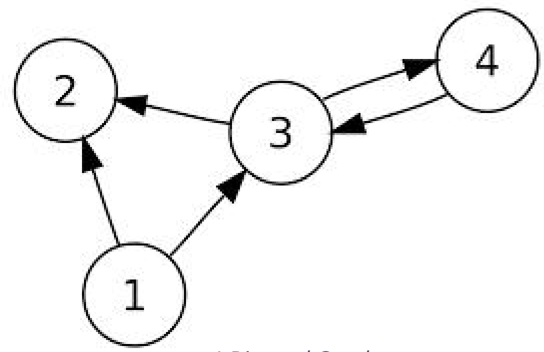
# Cycle Detection in Directed Graphs

**GITHUB REPOSITORY:**

https://github.com/HamzaButt22/Analysis-Of-Algorithm/tree/main/Project%20-%20Cycle%20Detection%20In%20Directed%20Graph

## Introduction

In directed graphs, cycle detection is a fundamental problem. A *cycle* exists if a path leads back to the same node. This has implications in deadlock detection, task scheduling, and compiler design. The most common methods to detect cycles are:



1 Directed Graph

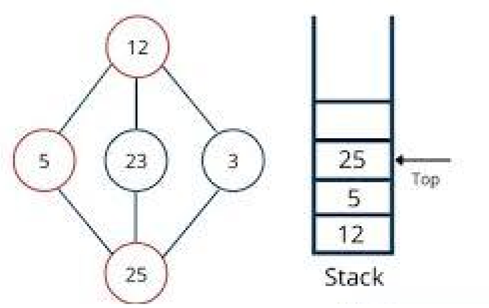
- **Depth-First Search (DFS) with recursion stack**
- **Breadth-First Search (BFS) using Kahn's Algorithm (Topological Sorting)**

---

1. **Cycle Detection Using DFS**

**Concept:**

- DFS traverses the graph by going deep along each branch before backtracking.
- We maintain a **recursion stack** to track nodes in the current DFS path.
- If during traversal we revisit a node that's already in the recursion stack, a **cycle exists**.




**Time Complexity:**

- **O (V + E)** where V is the number of vertices and E is the number of edges.
- Every node and edge are visited once.

**Space Complexity:**

- **O(V)** for visited and recursion stack arrays.

**Pseudocode Summary:**

```
DFS (node):
  mark node as visited
  add node to recursion stack
  for each neighbor:
    if not visited:
      DFS (neighbor)
    else if neighbor is in recursion stack:
      cycle exists
  remove node from recursion stack
```

**Example Output:**

For input edges:

{0→1}, {0→2}, {1→2}, {2→0}, {2→3}

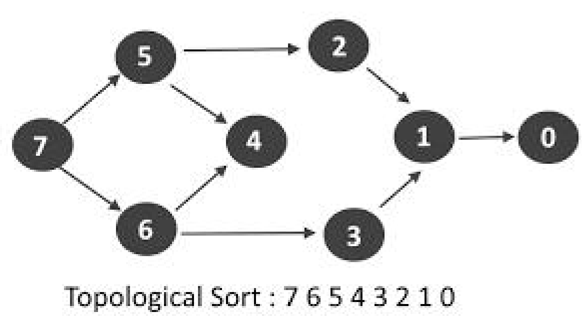
The DFS detects a **cycle** (0→1→2→0).

---

2. **Cycle Detection Using Kahn's Algorithm (BFS)**

**Concept:**

- Based on **topological sorting**.
- A graph with a cycle **cannot** have a valid topological order.
- Count in-degrees of each node.
- Remove nodes with in-degree 0 iteratively.
- If all nodes are not removed (i.e., count ≠ V), a **cycle exists**.




**Time Complexity:**

- **O (V + E)** similar to DFS. Each node and edge are processed once.

**Space Complexity:**

- **O(V)** for in-degree array and queue.

**Pseudocode Summary:**

```
Kahn (V, adj):
  Compute in-degree of all nodes
  Add nodes with in-degree 0 to queue
  while queue is not empty:
    remove node from queue
    for each neighbor:
      decrement in-degree
      if in-degree becomes 0:
        add to queue
  if total processed nodes! = V:
    cycle exists
```

**Example Output:**

For the same input edges:

{0→1}, {0→2}, {1→2}, {2→0}, {2→3}

Kahn's algorithm also detects a **cycle** due to incomplete topological sorting.

---

**Comparison Table**

| **Feature** | **DFS Method** | **Kahn's Algorithm (BFS)** |
|---|---|---|
| Technique | Recursive DFS with backtracking | BFS with in-degree processing |
| Suitable For | Simple recursive cycle check | Detecting topological order |
| Space Complexity | O(V) | O(V) |
| Time Complexity | O(V + E) | O(V + E) |
| Detects All Cycles? | Yes | Yes |

| | | |
|---|---|---|
| Graph Type | Directed only | Directed only |

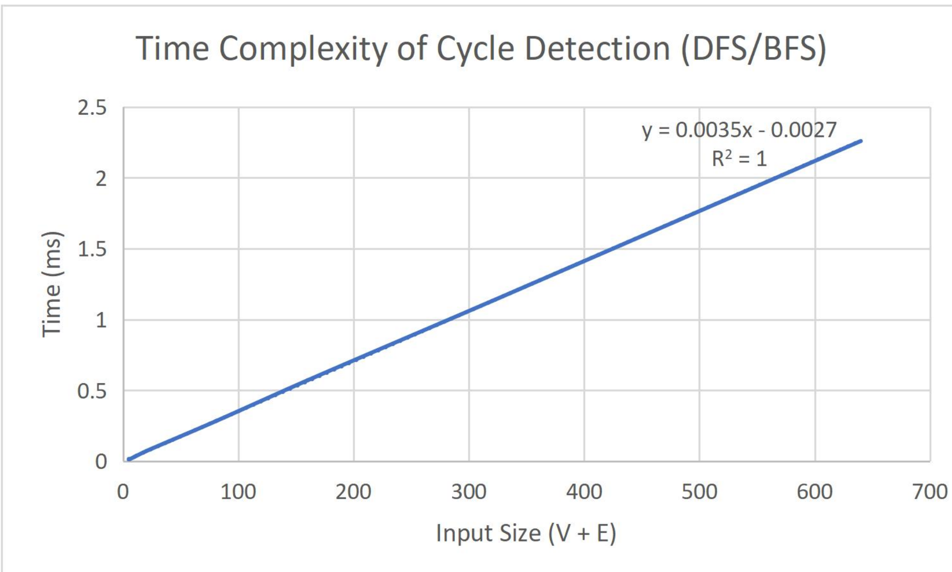
---

**Graphical Time Complexity Representation**

Let:

- **V** = number of vertices
- **E** = number of edges

Both DFS and BFS exhibit **linear time complexity**, i.e., **O (V + E)**.




---

**Conclusion**

- Both **DFS** and **BFS (Kahn's Algorithm)** are efficient for cycle detection in directed graphs.
- **DFS** is easier to implement recursively, while **BFS (Kahn's)** is preferable when topological sorting is needed.
- Both operate in **O (V + E)** time, optimal for sparse graphs.