# Kahn's algorithm for Topological Sorting

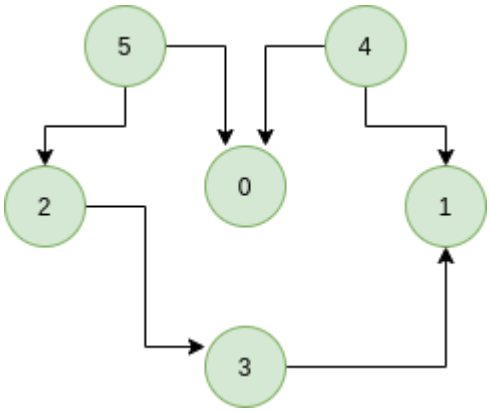
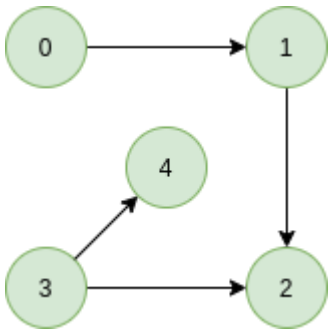
geeksforgeeks.org/dsa/topological-sorting-indegree-based-solution

April 4, 2016

Given a **Directed Acyclic Graph** having **V** vertices and **E** edges, your task is to find any **Topological Sorted order** of the graph.

**Topological Sorted order**: It is a linear ordering of vertices such that for every directed edge **u -> v**, where vertex **u** comes before **v i**n the ordering.

**Example**:

> **Input:** V=6 , E = [[2,3], [3,1], [4,0], [4,1], [5,0], [5,2]]
>
> 
> 
>
> **Output:** 4 5 2 0 3 1
> **Explanation**: In the above output, each dependent vertex is printed after the vertices it depends upon.
>
> **Input**: V=5 , E=[[0,1], [1,2[, [3,2], [3,4]]
>
> 
> 
>
> **Output**: 0 3 4 1 2
> **Explanation**: In the above output, each dependent vertex is printed after the vertices it depends upon.

## Kahn's Algorithm for Topological Sorting:

> **Kahn's Algorithm** works by repeatedly finding vertices with no incoming edges, removing them from the graph, and updating the incoming edges of the vertices connected from the removed removed edges. This process continues until all vertices have been ordered.

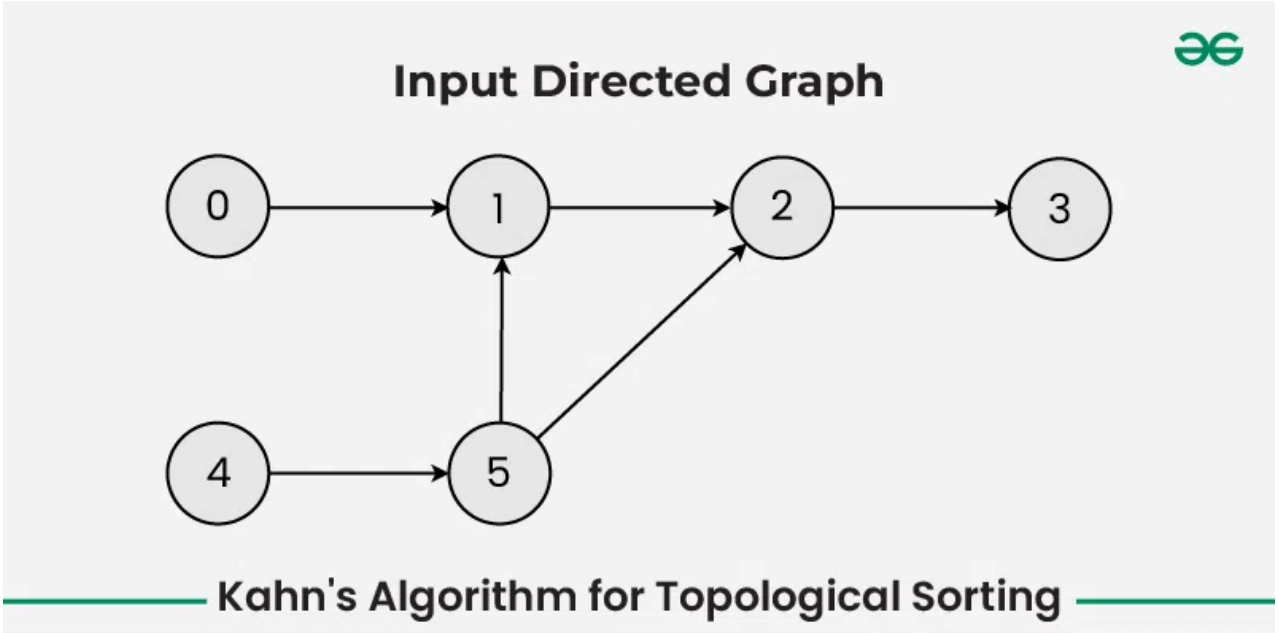
## Algorithm:

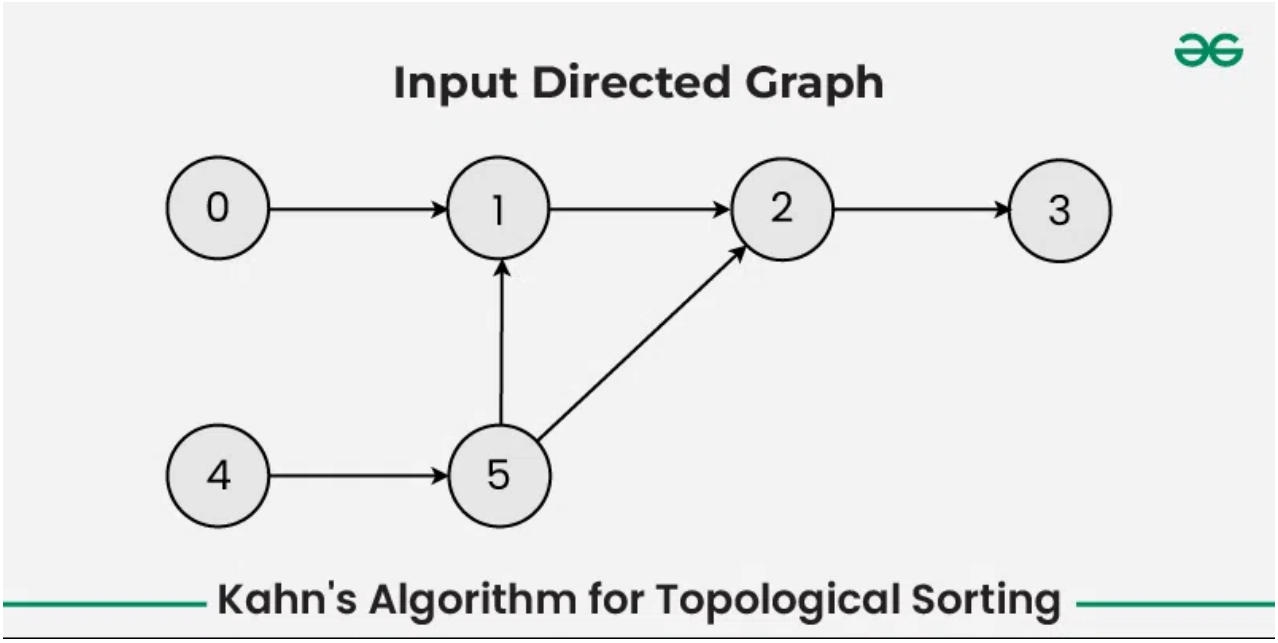
- Add all nodes with in-degree **0** to a queue.
- While the queue is not empty:
  - Remove a node from the queue.
  - For each outgoing edge from the removed node, decrement the in-degree of the destination node by **1**.
  - If the in-degree of a destination node becomes **0**, add it to the queue.
- If the queue is empty and there are still nodes in the graph, the graph contains a cycle and cannot be topologically sorted.
- The nodes in the queue represent the topological ordering of the graph.

## How to find the in-degree of each node?

> To find the **in-degree** of each node by initially calculating the number of incoming edges to each node. Iterate through all the edges in the graph and **increment** the **in-degree** of the **destinationnode** for each edge. This way, you can determine the in-degree of each node before starting the sorting process.

**Working of the above algorithm:**






Below is the implementation of the above algorithm.

Python

```
from collections import deque

# We mainly take input graph as a set of edges. This function is

# mainly a utility function to convert the edges to an adjacency

# list

def constructadj(V, edges):

    adj = [[] for _ in range(V)]

    for u, v in edges:

        adj[u].append(v)

    return adj


# Function to return list containing vertices in Topological order

def topologicalSort(V, edges):

    adj = constructadj(V, edges)

    indegree = [0] * V


    # Calculate indegree of each vertex

    for u in range(V):
```

```
        for v in adj[u]:
            indegree[v] += 1

    # Queue to store vertices with indegree 0
    q = deque([i for i in range(V) if indegree[i] == 0])

    result = []
    while q:
        node = q.popleft()
        result.append(node)

        for neighbor in adj[node]:
            indegree[neighbor] -= 1
            if indegree[neighbor] == 0:
                q.append(neighbor)

    # Check for cycle
    if len(result) != V:
        print("Graph contains cycle!")
        return []

    return result

if __name__ == "__main__":
    V = 6
    edges = [[0, 1], [1, 2], [2, 3], [4, 5], [5, 1], [5, 2]]

    result = topologicalSort(V, edges)
    if result:
        print("Topological Order:", result)
```

C# JavaScript

**Output**

```
0 4 5 1 2 3
```

**Time Complexity:** O(V+E). The outer for loop will be executed V number of times and the inner for loop will be executed E number of times.
**Auxiliary Space:** O(V). The queue needs to store all the vertices of the graph.

> We do not count the adjacency list in auxiliary space as it is necessary for representing the input graph.

## Applications of Kahn's algorithm for Topological Sort:

- **Course sequencing:** Courses at universities frequently have prerequisites for other courses. The courses can be scheduled using Kahn's algorithm so that the prerequisites are taken before the courses that call for them.
- **Management of software dependencies:** When developing software, libraries and modules frequently rely on other libraries and modules. The dependencies can be installed in the proper order by using Kahn's approach.
- **Scheduling tasks:** In project management, activities frequently depend on one another. The tasks can be scheduled using Kahn's method so that the dependent tasks are finished before the tasks that depend on them.
- **Data processing:** In data processing pipelines, the outcomes of some processes may be dependent. The stages can be carried out in the right order by using Kahn's algorithm.
- **Circuit design:** In the creation of an electronic circuit, some components may be dependent on the output of others. The components can be joined in the right order by using Kahn's technique.