Topological Sorting (Kahn's BFS Based Algortihm)

Topological sorting for **D**irected **A**cyclic **G**raph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

For example, a topological sorting of the following graph is “5 4 2 3 1 0?. There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is “4 5 2 0 3 1". The first vertex in topological sorting is always a vertex with an in-degree of 0 (a vertex with no incoming edges).

Dash

All

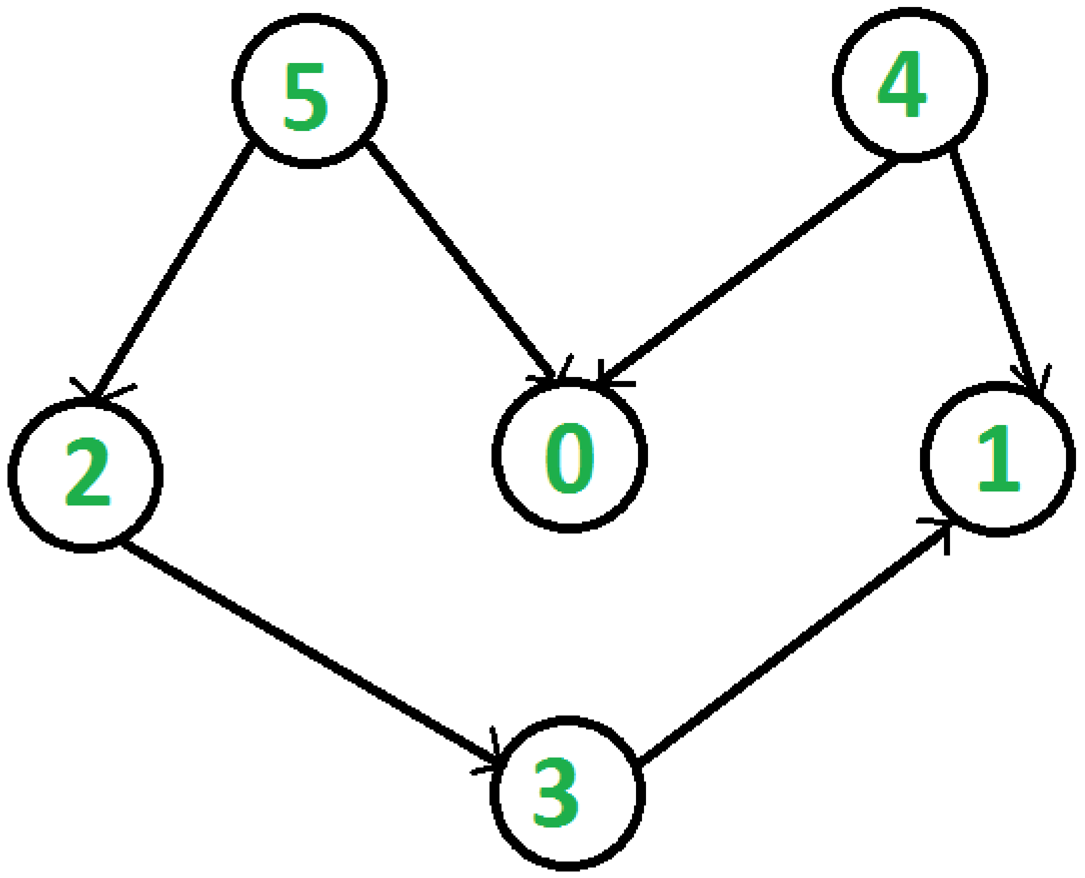
Articles

Videos

Problems

Quiz

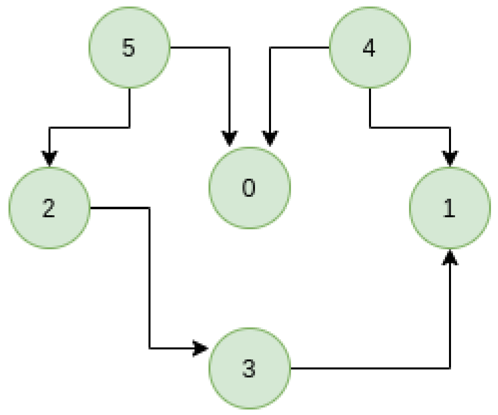
Contest



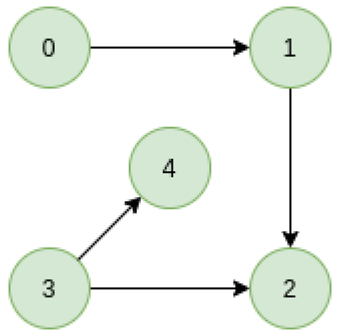
Let's look at a few examples with proper explanation,

Example:

> **Input:**



Output: 5 4 2 3 1 0
Explanation: The topological sorting of a DAG is done in a order such that for every directed edge uv, vertex u comes before v in the ordering. 5 has no incoming edge. 4 has no incoming edge, 2 and 0 have incoming edge from 4 and 5 and 1 is placed at last.
Input:



Output: 0 3 4 1 2
Explanation: 0 and 3 have no incoming edge, 4 and 1 has incoming edge from 0 and 3. 2 is placed at last.

A DFS based solution to find a topological sort has already been discussed.

Solution: In this article, we will see another way to find the linear ordering of vertices in a directed acyclic graph (DAG). The approach is based on the below fact:
A DAG G has at least one vertex with in-degree 0 and one vertex with out-degree 0.
Proof: There's a simple proof to the above fact that a DAG does not contain a cycle which means that all paths will be of finite length. Now let S be the longest path from u(source) to v(destination). Since S is the longest path there can be no incoming edge to u and no outgoing edge from v, if this situation had occurred then S would not have been the longest path
=> indegree(u) = 0 and outdegree(v) = 0

Intuition:

Topological sorting is a kind of dependencies problem so, we can start with the tasks which do not have any dependencies and can be done straight away or simply if we talk about in the term of node,

- We will always try to execute those nodes that have outdegree 0.
- Then after execution of all those 0 outdegrees, we will execute which are directly dependent on currently resolved tasks (currently resolved tasks' outdegrees will become 0 now) and so on will execute all other tasks.

We look closely we are doing these executions are done level-wise or in a Breadth-first search (BFS) manner. Similarly, we can also perform the same task for indegree=0.

Algorithm: Steps involved in finding the topological ordering of a DAG:
Step-1: Compute in-degree (number of incoming edges) for each of the vertex present in the DAG and initialize the count of visited nodes as 0.
Step-2: Pick all the vertices with in-degree as 0 and add them into a queue (Enqueue operation)

Step-3: Remove a vertex from the queue (Dequeue operation) and then.

1. Increment the count of visited nodes by 1.
2. Decrease in-degree by 1 for all its neighbouring nodes.
3. If the in-degree of neighbouring nodes is reduced to zero, then add it to the queue.

Step 4: Repeat Step 3 until the queue is empty.
Step 5: If the count of visited nodes is **not** equal to the number of nodes in the graph then the topological sort is not possible for the given graph.
How to find the in-degree of each node?
There are 2 ways to calculate in-degree of every vertex:

1. Take an in-degree array which will keep track of
   Traverse the array of edges and simply increase the counter of the destination node by 1.

```
for each node in Nodes
    indegree[node] = 0;
for each edge(src, dest) in Edges
    indegree[dest]++
```

1. Time Complexity: O(V+E)
2. Traverse the list for every node and then increment the in-degree of all the nodes connected to it by 1.

```
    for each node in Nodes
        If (list[node].size()!=0) then
        for each dest in list
            indegree[dest]++;
```

1. Time Complexity: The outer for loop will be executed V number of times and the inner for loop will be executed E number of times, Thus overall time complexity is O(V+E).
   The overall time complexity of the algorithm is O(V+E)

Below is the implementation of the above algorithm. The implementation uses method 2 discussed above for finding in degrees.

C++ Java

```java
// A Java program to print topological
// sorting of a graph using indegrees
import java.util.*;

// Class to represent a graph
class Graph {
    // No. of vertices
    int V;

    // An Array of List which contains
    // references to the Adjacency List of
    // each vertex
    List<Integer> adj[];
    // Constructor
    public Graph(int V)
    {
        this.V = V;
        adj = new ArrayList[V];
        for (int i = 0; i < V; i++)
            adj[i] = new ArrayList<Integer>();
    }

    // Function to add an edge to graph
    public void addEdge(int u, int v)
    {
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
        adj[u].add(v);
    }
    // prints a Topological Sort of the
    // complete graph
    public void topologicalSort()
    {
        // Create a array to store
        // indegrees of all
        // vertices. Initialize all
        // indegrees as 0.
        int indegree[] = new int[V];

        // Traverse adjacency lists
        // to fill indegrees of
        // vertices. This step takes
        // O(V+E) time
        for (int i = 0; i < V; i++) {
            ArrayList<Integer> temp
                = (ArrayList<Integer>)adj[i];
            for (int node : temp) {
                indegree[node]++;
            }
        }

        // Create a queue and enqueue
        // all vertices with indegree 0
        Queue<Integer> q
            = new LinkedList<Integer>();
        for (int i = 0; i < V; i++) {
            if (indegree[i] == 0)
                q.add(i);
        }

        // Initialize count of visited vertices
        int cnt = 0;

        // Create a vector to store result
        // (A topological ordering of the vertices)
```

Get 90% Refund!
Courses Tutorials Jobs Practice Contests

Dash

All

Articles

Videos

Problems

Quiz

Contest

```
                // (or perform dequeue)
                // and add it to topological order
                int u = q.poll();
                topOrder.add(u);

                // Iterate through all its
                // neighbouring nodes
                // of dequeued node u and
                // decrease their in-degree
                // by 1
                for (int node : adj[u]) {
                    // If in-degree becomes zero,
                    // add it to queue
                    if (--indegree[node] == 0)
                        q.add(node);
                }
                cnt++;
            }

            // Check if there was a cycle
            if (cnt != V) {
                System.out.println(
                    "There exists a cycle in the graph");
                return;
            }

            // Print topological order
            for (int i : topOrder) {
                System.out.print(i + " ");
            }
        }
    }
    // Driver program to test above functions
    class Main {
        public static void main(String args[])
        {
            // Create a graph given in the above diagram
            Graph g = new Graph(6);
            g.addEdge(5, 2);
            g.addEdge(5, 0);
            g.addEdge(4, 0);
```

```
                g.addEdge(4, 1);
                g.addEdge(2, 3);
                g.addEdge(3, 1);
                System.out.println(
                    "Following is a Topological Sort");
                g.topologicalSort();
            }
        }
```

Output

```
Following is a Topological Sort of
4 5 2 0 3 1
```

Complexity Analysis:

- **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. So the space required is O(V)

The above topological sorting algorithm will only work when the graph doesn't contain any cycle. Although we can modify the above algorithm to detect cycle in the graph.

If you are facing any issue on this page. Please let us know.

Dash

All

Articles

Videos

Problems

Quiz

Contest