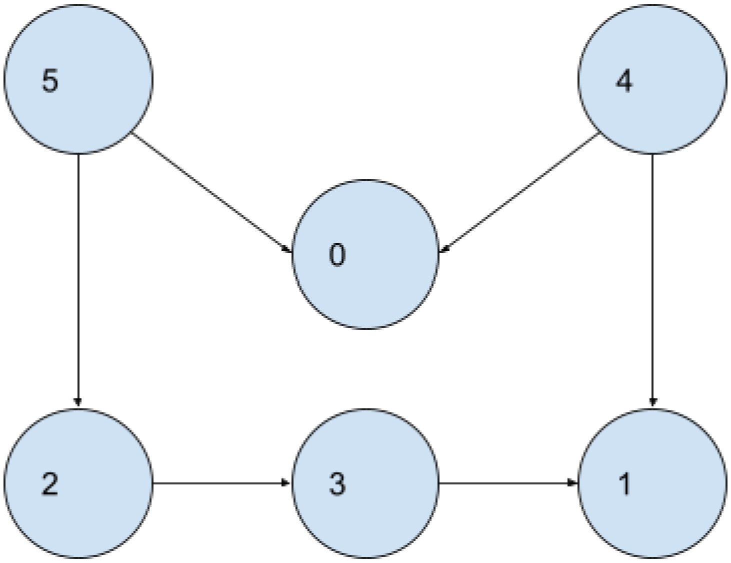
MEDIUM

Kahn’s Algorithm | Topological Sort Algorithm (USING BFS)

Intuition



Definition : Linear Ordering of vertices such that if there is an edge between U and V before V in that ordering.

DAG - Directed Acyclic Graph , directed graph that doesn't have any cycle

Eg.

5 -> 0
4 -> 0
5 -> 2
2 -> 3
3 -> 1
4 -> 1

5 appears before 0
4 appears before 0
5 appears before 2
2 appears before 3
3 appears before 1
4 appears before 1

It gives us two orders possible

Order 1 : 5 4 2 3 1 0

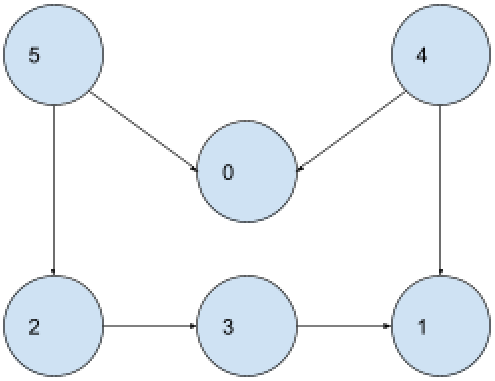
Order 2 : 4 5 2 3 1 0

Why only in DAG ?

If we take an undirected graph then 1-2 then 2-1 is also there which is not possible hence only directed graph.

Now if we have a directed graph with a cycle then also a condition arises.
Eg. path 1-2-3-1
1 before 2
2 before 3
3 before 1 [not possible]

Eg.



Visited = [1 1 1 1 1 0]

Adjacency List :

0 - [ ]
1 - [ ]
2 - [3]
3 - [1]
4 - [0, 1]
5 - [0, 2]

To implement the Kahn’s Algorithm using the Breadth first search and modifying it

We will be implementing the indegree array

Indegree will contain the number of incoming edges

Eg.

Indegree = [2, 2, 1, 1, 0, 0]

U appears before V according to toposort

So we can say that

- 5 and 4 do not have any indegree and has no one coming before them therefore they can be placed in the start
- Insert them to the queue first
- pop(4) now we have 0 and 1 and thus their indegree will be reduced by 1 from both
- Indegree = [1, 1, 1, 1, 0, 0] // 4 connects to 0 and 1
- Pop (5) now we will again repeat Indegree = [0,1,0,1,0,0]
- 0,2 now can be pushed into the queue
- pop(0)
- pop(2) - reduce Indegree = [0, 1, 0, 0, 0, 0]
- push(3)
- pop(3) - reduce indegree = [0, 0, 0, 0, 0, 0]

Output : 4 5 0 2 3 1

Approach

- Create an indegree vector having initialization of 0 for n elements
- Initially assign the indegree of all elements to the indegree vector
- Declare
    - An empty queue
    - A result vector containing the topological order
- Traverse until the queue becomes empty :
    - Extract the first element of the queue
    - Pop the first element of the queue
    - Insert the first element in the result vector
    - Traverse for all adjacent elements :
        - Reduce the indegree of elements by 1
        - If indegree of any element becomes 0 push it to queue
- Return the result vector

Function Code

```
vector<int> topoSort(int n, vector<int> adj[])
      {
          // creating a vector for indegree having n elements
        vector<int> indegree(n, 0);

        // assigning the indegree of every element at initialization
        for (int i = 0; i < n; i++) {
            for (int j : adj[i]) {

                indegree[j] += 1;
            }
        }

        // declaring an empty queue
        queue<int> q;

        // traversing for all graph components and pushing the elements
with the 0 indegree to the queue
        for (int i = 0; i < n; i++) {

            if (indegree[i] == 0) {
                q.push(i);
            }
        }

        // vector for result
        vector<int> result;

        // traversing until the queue becomes empty
        while (!q.empty()) {
            // extracting the first node
            int node = q.front();

            // popping the first node
            q.pop();

            // inserting the first node to the result vector
            result.push_back(node);

            // traversing for all the adjacent elements
            for (int i : adj[node]) {.
```

```
                indegree[i] -= 1;

                   if (indegree[i] == 0) {
                    q.push(i);
                }
            }
        }

        return result;
```

Time Complexity

O(V+E)