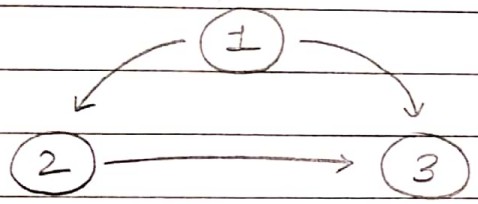
21/06/2023

Topological sort

Linear ordering of vertices such that for every edge $u \to v$, u comes before v in that ordering.

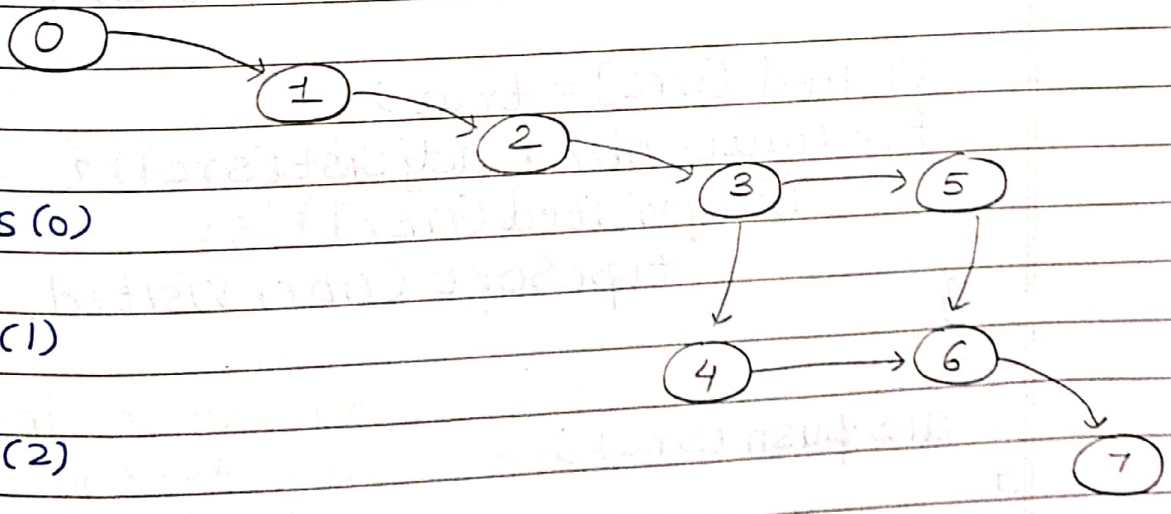


$1 \to 3$
$2 \to 3$
$1 \to 2$
} Adjacency list

Valid topological sort ⇒ $1 \to 2 \to 3$
Invalid topological sort ⇒ $2 \to 1 \to 3$

Note→ Nodes with no dependency are printed.
Topological sort can be applied only on DAG i.e directed acyclic graph.

Ex→



dfs(0)
↓
dfs(1)
↓
dfs(2)
↓
dfs(3) → dfs(5)
↓
dfs(4) → dfs(6) → dfs(7)

Stack → While returning, simply push that node

0
1
2
3
5
4
6
7

Now fetch elements from stack and we will get a valid topological sort.

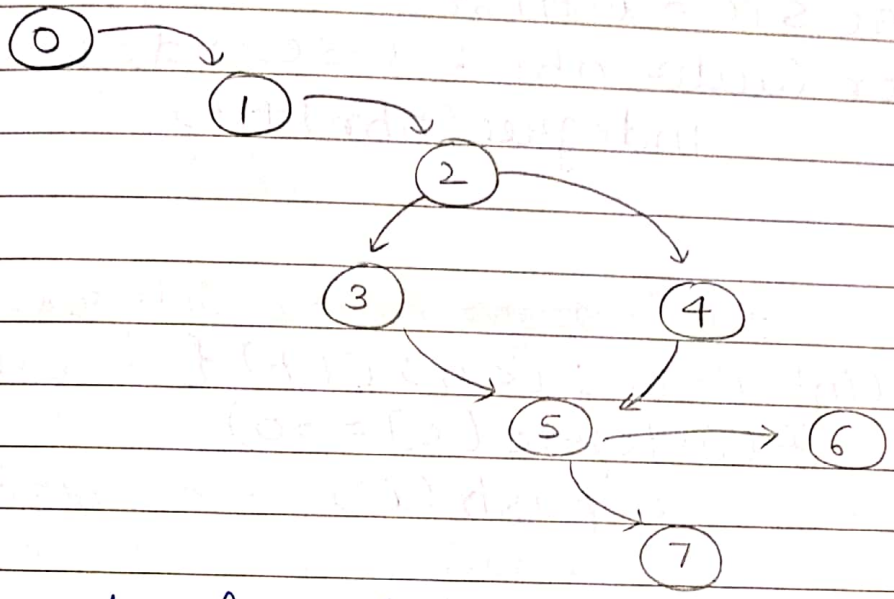
0 → 1 → 2 → 3 → 5 → 4 → 6 → 7

Code

```
void topoSort (int src, unordered_map <int, bool> & visited, stack <int> & ans) {

    visited [src] = true;
    for (auto nbr : adjList [src]) {
        if (!visited [nbr])
            topoSort (nbr, visited, ans);
    }
    // While returning, push node in the stack.
    ans.push (src);
}
```

Topological sort using BFS.
This is known as Kahn's algorithm



The nodes have indegree = 0, simply push in queue.

queue → {0} (Remove 0 from graph)
queue → {0, 1} (Remove 1 from graph)
queue → {0, 1, 2} (Remove 2 from graph)
queue → {0, 1, 2, 4, 3} (Remove 4 & 3 from graph)
queue → {0, 1, 2, 4, 3, 5} (Remove 5 from graph)
queue → {0, 1, 2, 4, 3, 5, 6, 7} (Remove 6 and 7 from graph)

Hence topological sort is
0 → 1 → 2 → 4 → 3 → 5 → 6 → 7.

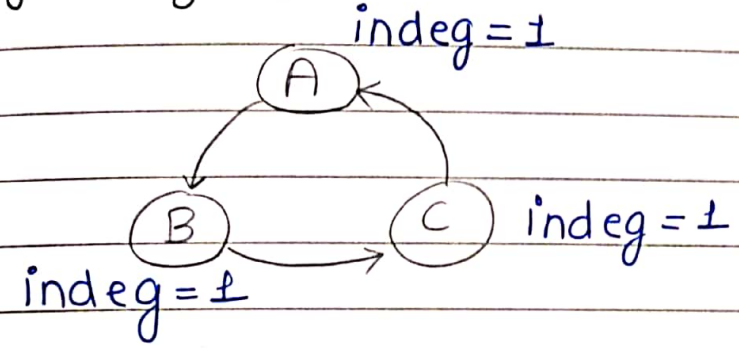
Code

```
void topoSort (int n, vector<int>& ans) {
    queue<int> q;
    unordered_map<int, int> indegree;
    //Calculate indegree
    for (auto i : adjList) {
```

```
        int src = i.first;
        for (auto nbr : i.second) {
            indegree[nbr]++;
        }
    }
    // Put nodes in queue having indegree = 0
    for (int i=0; i<n; i++) {
        if (indegree[i] == 0)
            q.push(i);
    }
    // BFS logic
    while (!q.empty()) {
        int frontNode = q.front();
        q.pop();
        ans.push_back(frontNode);
        for (auto nbr : adjList[frontNode]) {
            // To remove nodes from graph
            indegree[nbr]--;
            // Check for 0 indegree again
            if (indegree[nbr] == 0) {
                q.push(nbr);
            }
        }
    }
}
```

Note → visited is not required in case of connected graphs as we are checking for indegree before & then pushing in the queue. Visited is not required for disconnected graphs also as we have traversed the full graph and calculated the indegree.

Why topological sort can't be applied on cyclic graphs ?



As here indeg = 1 for all nodes & indeg = 0 for no node and hence we are not able to print all nodes of topological sort.

Cycle detection using BFS.
If we are able to find valid topological sort, then cycle is absent. If no valid topological sort, then simply say cycle is present.

```
main() →
    if (ans.size() == n)      → no. of nodes of graph
        cout << "No cycle";
    else
        cout << "cycle present";
```