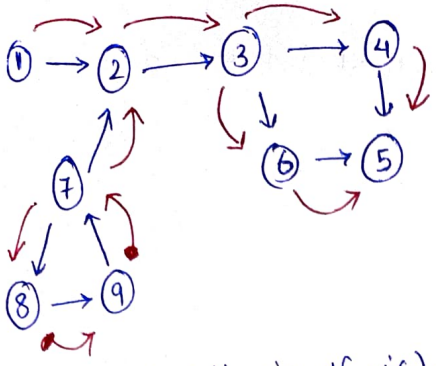
- Cycle Detection in Directed graph using DFS



Here, the undirected graph approach will not work because lets suppose we traverse from 1 to 2, 3, 4 and 5 and from 3 to 6 and 5 Now, at this point (6) as 5 is not a parent of 6 and also 5 is already visited, so it will return true but in this example there is no cycle in 5.

Approach:

- We have to create 2 arrays for marking visited node.
  - First array for keeping track of visited.
  - Second array for keeping track of visited nodes but in a particular recursion call by which we can say if node is visited in both the array, just return true otherwise NO.

```
bool check(node, adj, vis, dfs_vis) {
    vis[node] = 1;
    dfs_vis[node] = 1;
    for(auto it : adj[node])
        if(!vis[it]) {
            if(check(it, ...)) return true;
        }
        else if(vis[it] && dfs_vis[it])
            return true;
    dfs_vis[node] = 0;
    return false;
}
bool isCycle(V, adj[]) {
    vector<int> vis(V,0), dfs_vis(V,0);
    for(auto it : adj)
        if(!vis[it])
            if(check(it,
    for(i → V)
        if(!vis[i])
            if(check(i, ...)) return true;
    return false;
}
```

- Topological Sort (Using DFS)

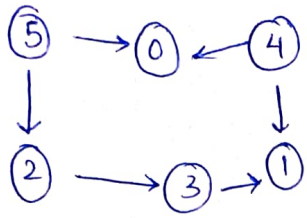
↳ Linear ordering of vertices such that if there is an edge u → v, u appears before v in that ordering.

↳ Graph should be Directed Acyclic Graph (DAG) to perform topological sort.

Intuition:

Suppose, u → v

it means u should appears beforev in answer, so in order to get this we have to push v in to the stack and the u. So if we pop out element u will be first one to out and then u.



Answer in topological sort
{5, 4, 2, 3, 1, 0}

adj. list

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

Approach:

- Perform DFS and visit all the nodes while marking it visited array. and a particular node's adjacent nodes are visited. then just ~~push~~ push particular node in stack.
- At last pop out all the nodes from stack. and push back to the vector.

```
void findtoposort (node, vis, st, adj) {
        vis [node] = 1;
        for (auto it : adj [node]) {
            if (! vis [it])
                findtoposort (it, vis, ..);
        }
        st.push (node);
    }
vector <int> topologicalsort (V, adj) {
        vector <int> vis(V, 0);
        stack <int> st;
        for(i → v)
            if (! vis[i])
                findtopsort (i, vis, st, adj);
        while (! st.empty) {
            ans.push_back( st.top());
            st.pop();
        }
        return ans;
    }
```

- **Topological sort (Using BFS)**

Approach: Firstly we will find the indegrees of all the 'V' nodes. Secondly, we will push the node in to the queue whose indegree is 0. And start popping out the elements decrementing the indegree of its adjacent nodes, and push the popped element into our answer array. & if the indegree of adjacent node gets 0, Just push it to the queue.

```
vector<int> findTopo (V, adj[]) {
    queue <int> q;
    vector <int> indeg(V,0), ans;
    for (i→v) if (v
    for (i→v)
        for (auto it :adj[i])
            indeg[it]++;
    for (i→v)
        if (indeg[i] == 0)
            q.push(i);
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        ans.push_back(node);
        for(auto it : adj[node]) {
            indeg[it]--;
            if (!indeg[it])
                q.push(it);
        }
    }
    return ans;
}
```

- **Cycle Detection in Directed graph (Using BFS)**

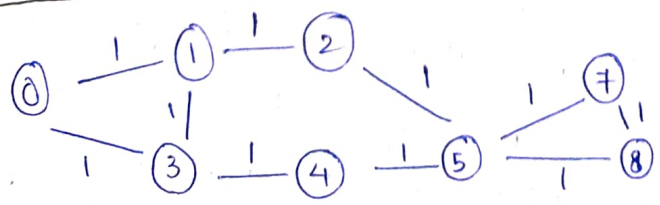
Just add a counter in Kahn's Algorithm and Atlast just check if the value of count is equal to 'v' vertices, there return false otherwise true.

```
int c=0;
int node = q.front();
q.pop();
c++;
_____
    _____
_____
if (c == v) return false;
return true;
```

Intuition:

If the given graph is DAG, then the topological sort of this graph will exists but if its not then we can return false.

- **Shortest Path in Undirected graph with Unit weights**



~~SET~~
src→0
dist]
{0,1,2,1,2,3,4,4}

Intuition: Create distance array that will store distance from 0 (~~Set~~ src).
assign all the values of distance by infinity (INT_MAX).
Now, take src node, and mark its distance 0 as src to src is 0 distance
and push src to the queue.
Now, pop out the node from queue, and check the distance of adjacent nodes with ~~p~~popped nodes and, if the distance with adjacent node is greater, just change the distance of adjacent node. and push it into the queue.
while q is not empty {

```
void shortestDist(adj[], V, src) {
    vector<int> dist(V, INT_MAX);
    queue<int> q;
    dist[src] = 0;
    q.push(src);
    while(!q.empty()) {
        int node = q.front();
        q.pop();
        for(auto it: adj[node]) {
            if(dist[node]+1 < dist[it]) {
                dist[it] = dist[node]+1;
            }
        }
    }
    return dist;
}
```