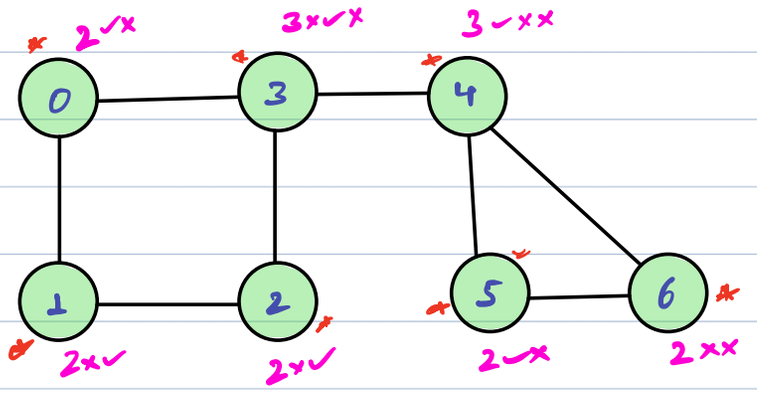
Today's agenda

- DFS
- No. of Connected Component +1
- Topological Sort +1

DFS → Depth first search Traversal

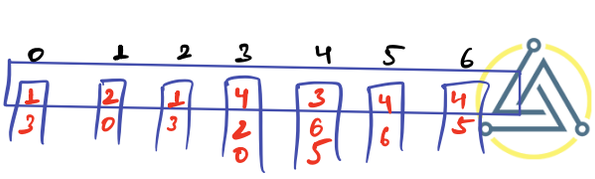


↳ A traversal on graph in any order.

//Pseudo Code

```
void dfs (list<list<Integer>>graph, boolean vis[v], int src){

    list<Integer> nbrs = graph.get(src);

    for(int i=0; i<nbrs.size(); i++){
        int v = nbrs.get(i);
        if(vis[v] == false){
            vis[v] = true;
            dfs(graph, vis, v);
        }
    }

}
```

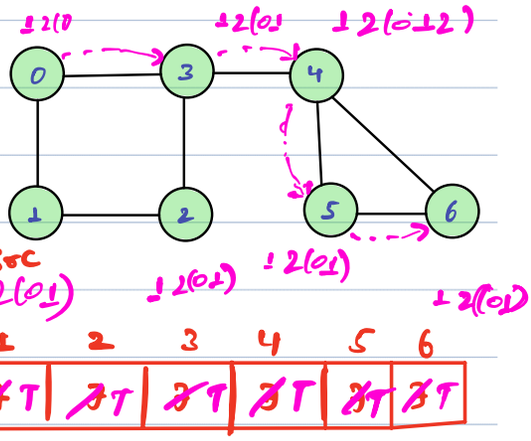
graph



main c1{

3

```
void dfs (list<list<Integer>>graph, boolean vis[], int src){
1   list<Integer> nbrs = graph.get(src);
2   for(int i=0; i<nbrs.size();i++){
      int v= nbrs.get(i);
      if(vis[v]==false){
        vis[v]=true;
3       dfs(graph, vis, v);
      }
    }
3
```

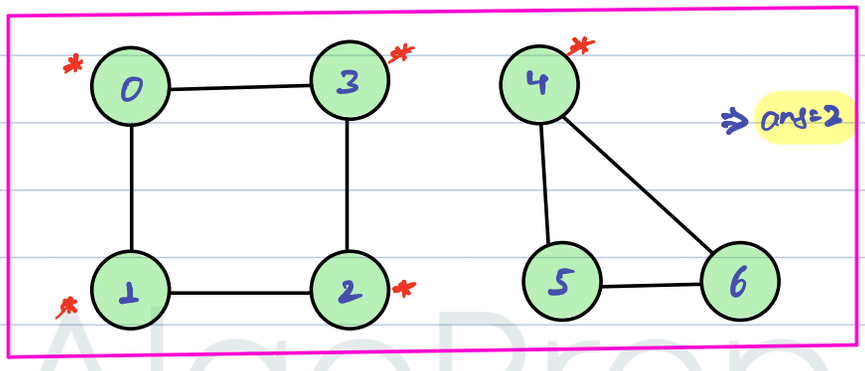


Q) Connected Components

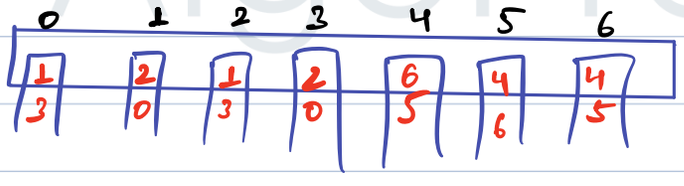
↳ Given undirected graph, find no. of Connected Components.

Note: A Component is said to be Connected, if from every node we can visit all nodes inside that Componend.

Ex:



graph



//Pseudo code → multi function dfs

```
int main() {
    int n = // no. of nodes     int m = // no. of edges
    // construction
    List<List<Integer>> graph = new ArrayList<>();
    int ans = 0;
    boolean[] vis = new boolean[n];
    for (int i = 0; i < n; i++) {
        if (vis[i] == false) {
            vis[i] = true;
            dfs(graph, vis, i);
            ans++;
        }
    }
    return ans;
}

void dfs(List<List<Integer>> graph, boolean vis[n], int src) {

    List<Integer> nbrs = graph.get(src);

    for (int i = 0; i < nbrs.size(); i++) {
        int v = nbrs.get(i);
        if (vis[v] == false) {
            vis[v] = true;
            dfs(graph, vis, v);
        }
    }
}
```

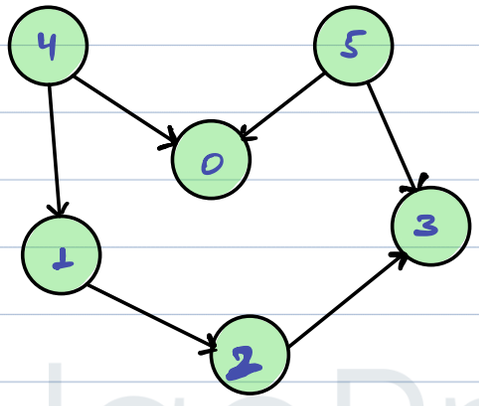
Break 9:32 PM

→ Kahn's algo

Q) Topological Sort

↳ Give the linear ordering of graph such that for every directed edge (u,v) vertex u comes before v in the ordering. { DAG → Directed, Acyclic graph only }

Ex:



TS: 1 3 2 0 4 5 ✗✗

TS: 4 0 5 1 3 2 ✗✗

TS: 4 5 0 1 2 3 ✓

TS: 5 4 0 1 2 3 ✓

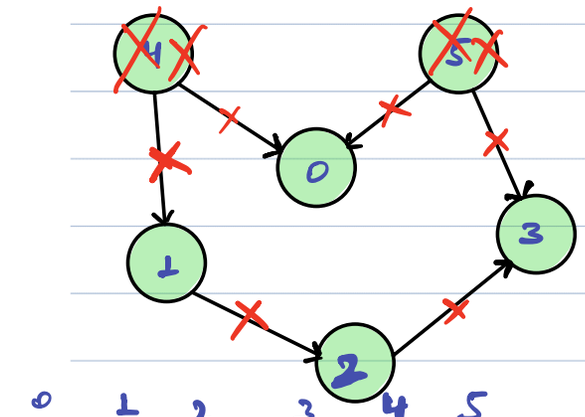
} multiple topological sort of same graph is possible.

// Kahn's algo

Count of incoming edges for each node.

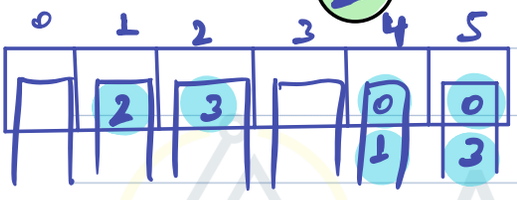
indegree:

0	1	2	3	4	5
~~2~~ 1 0	~~1~~ 0	~~1~~ 0	~~2~~ 1 0	0	0



q: ~~4~~ ~~5~~ ~~1~~ ~~0~~ ~~2~~ ~~3~~

rem = 3



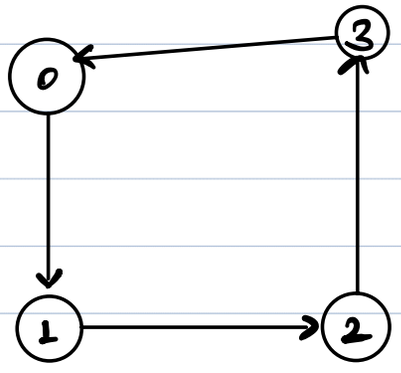
TS: 4 5 1 0 2 3

// Pseudo Code

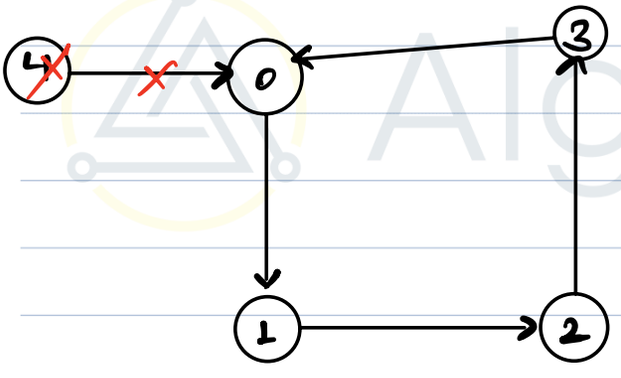
```
void topological_sort (List<List<Integer>> graph, int n){
    int[] indegree = new int[n];

    for (int u=0; u<n; u++){
        List<Integer> nbrs = graph.get(u);
        for (int v: nbrs){
            indegree[v]++;
        }
    }
```

```
Queue<Integer> q = new LinkedList<>();

    for(int i=0; i<n; i++) {
        if(indegree[i]==0) { q.add(i); }
    }

    int count=0;
    while(q.size()>0){
        int rem = q.remove();
        S.O.P(rem); → count++;

        List<Integer> nbrs = graph.get(rem);
        for(int v: nbrs)
            indegree[v]--;
            if(indegree[v]==0){ q.add(v); }
    }
if(count==n) { S.O.P("acyclic"); }
    else { S.O.P("cyclic"); }
}
```

(I) Why not cyclic graph?



	0	1	2	3
indegree:	1	1	1	1



	0	1	2	3	4
indegree:	~~1~~ 4	1	1	1	0

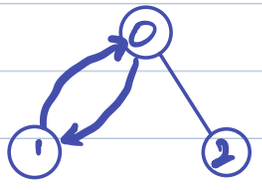
q: [4]

rem: 4

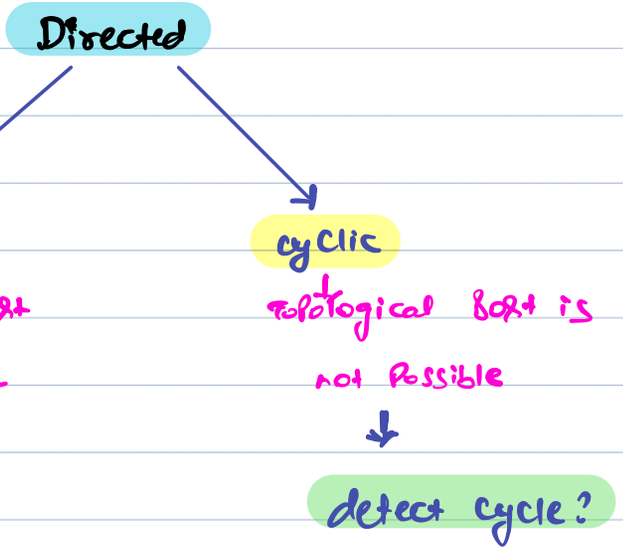
4

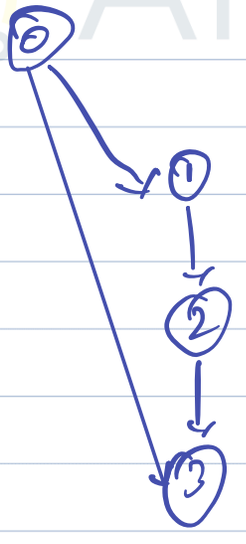
In cyclic graph, dependency is also cyclic.
No node to start the topological sort.

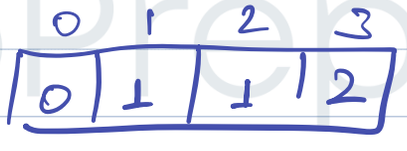
(II) Why not undirected graph? → Same issue as above.



→ if "DAG" term is in the Problem Statement, most likely first step is going to be topological sort.







0 1 3 2

HW

2) Course Schedule

Ls Given N courses that you have to take. you are also given Prerequisite in the form [xi, yi] indicating that you must take Course yi before xi. Come up with a valid ordering of all the courses that you are taking.

Ex: N=5 {1,0} {2,0} {3,1} {3,2} {3,0}