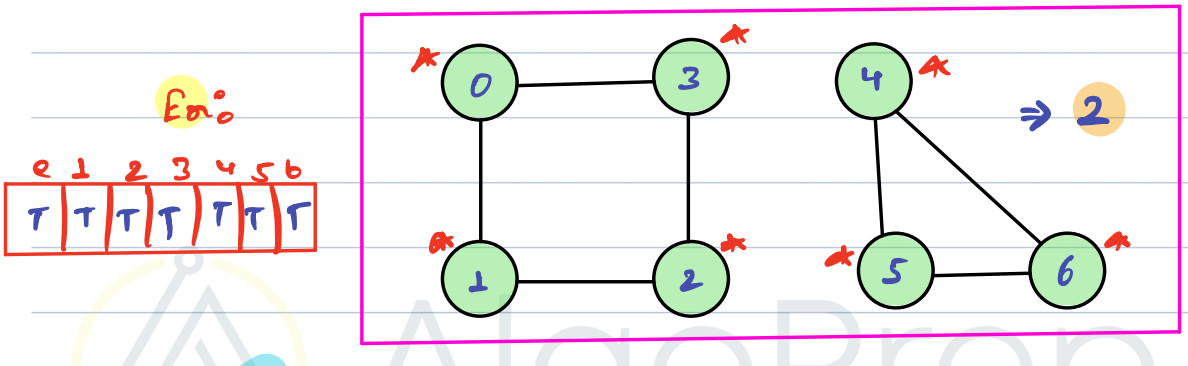
Today's agenda

- No. of Connected Component
- Topological Sort

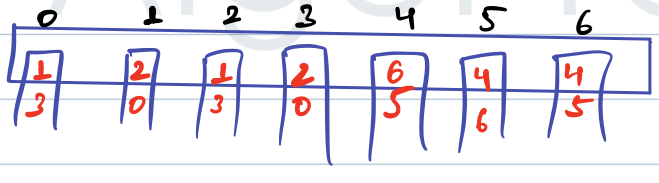
Q) Connected Components

↳ 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 Componerd.



graph



//Pseudo code

```
//multi function dfs
int main() {
    int n =          int m =
    List<List<Integer>> graph = Construction();

    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 v: nbrs) {
        if(vis[v] == false) {
            vis[v] = true;
            dfs(graph, vis, v);
        }
    }

}
```

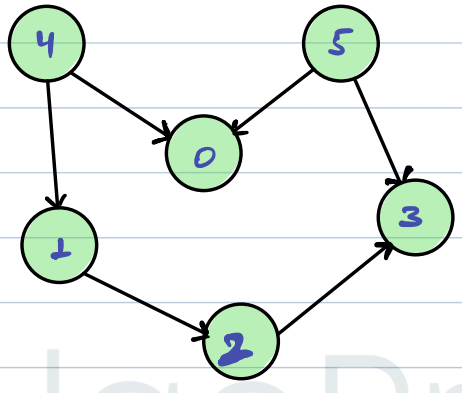
T.C: O(V+E)

S.C: O(V)

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 ✗✗

These can be multiple topological sort of same graph.

TS: 4 5 0 1 2 3 ✓✓

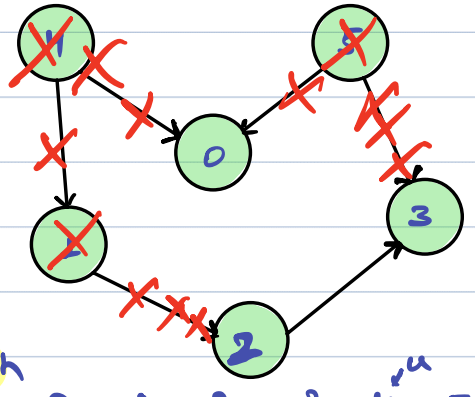
TS: 5 4 0 1 2 3 ✓✓

// Kahn's algo

no. of incoming edge of a node.

indegree:

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



q: 4 5 1 0 2 3

graph

0	1	2	3	4	5
	2	3		1 → 0	3 → 0

Topological sort:

4 5 1 0 2 3

// Pseudo code

```
void topologicalSort (List<List<Integer>> graph,
                      int v, int m) {

    int[] indegree = new int[v];

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

```
Queue<Integer> q;
```

T.C: O(V+E)

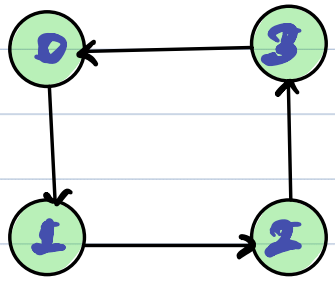
S.C: O(V)

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

while (q.size() >0){
    int rem = q.remove();
    s.o.p(rem);

List<Integer> nbrs = graph.get(rem);
    for (int v: nbrs) {
        indegree[v]--;
        if(indegree[v]==0){
            q.add(v);
        }
    }
}
}
```

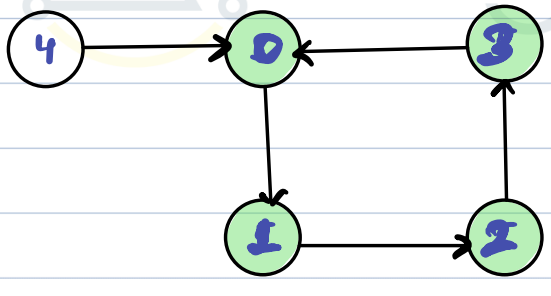
① what happens if graph is cyclic?



	0	1	2	3
indegree :	1	1	1	1

4 ⟶ ✔

4 < < ✔



	0	1	2	3	4
indegree :	1 → 2	1	1	1	0

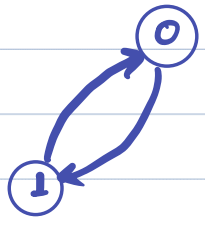
q: ~~4~~

size=4

4

→ in cyclic graph, dependency is also cyclic. nowhere to start from.

(11) why not undirected graph?



→ Same issue as cyclic graph.

Q) Detect cycle in directed graph.

directed graph

- acyclic
  - topological sort is valid.
- cyclic
  - you can't complete the topological ordering.

//Pseudo code

```
void topologicalSort(List<List<Integer>> graph,
                     int n, int m) {

    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;
```

T.C: O(V+E)
S.C: O(V)

```
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();
    ~~sop(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"); }
```

13