- Topological Order → 2 methods
- D.S.U. [Disjoint Set Union]
- Application for D.S.U.

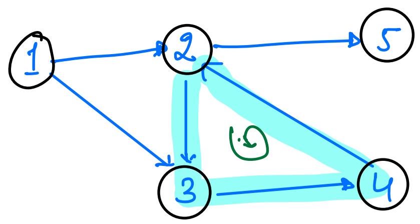
$$\begin{tikzcd} & & \text{Greedy.} \arrow[d] \\ \text{functions} \arrow[r] \arrow[urr, bend left] & \text{Recursion} \arrow[r] \arrow[d] & \underline{\text{D.P.}} \\ & \text{Backtracking} \arrow[ur] & \end{tikzcd}$$

$$\left[\begin{array}{c} \text{functions} \\ \downarrow \\ \text{Recursion} \\ \downarrow \\ \text{Backtracking} \\ \downarrow \\ \text{Greedy} \\ \downarrow \\ \text{D.P.} \end{array}\right]$$

Q: Given N courses with pre-requisite of each course. Check if it is possible to finish all the courses.

Eg: $N = 5$.

x	is a pre-requisite of →
1 →	2, 3
2 →	3, 5
3 →	4
4 →	2



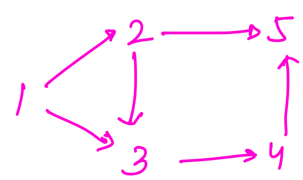
ans = false.

Solution → if graph is cyclic ⟹ ans = false
otherwise, ans = true.

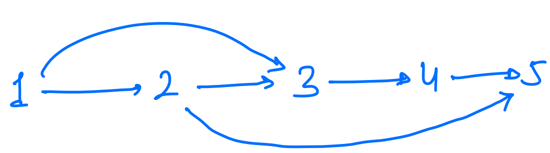
Topological Order / Sort.

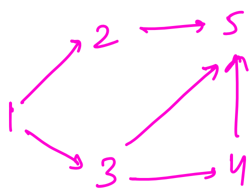
Linear ordering of nodes such that if there is an edge from i to j then i should be on left side of j. $(i < j)$

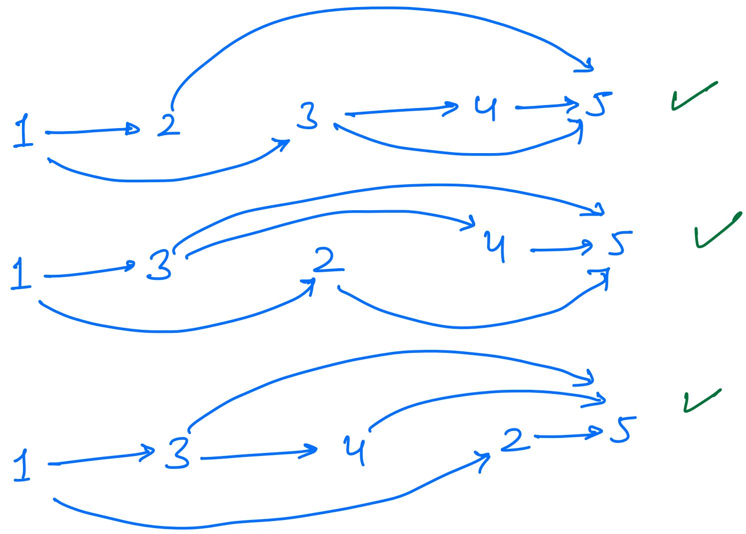
i.e. → Directed Acyclic Graph (D.A.G)



1, 2, 3, 4, 5



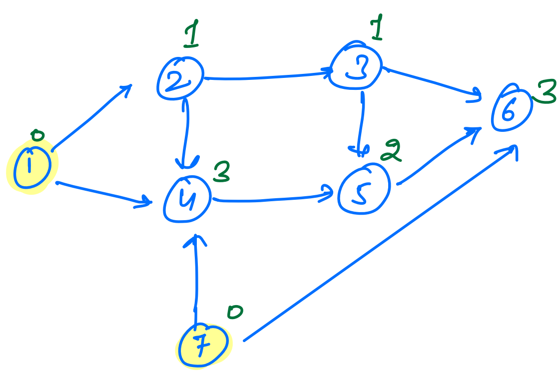




∴ Multiple topo-logical orders are possible.

Find Topological Order

① left to right.



steps → ① find indegree of all the nodes.

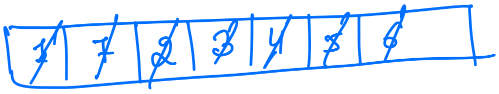
$\forall i$, in[i] = 0

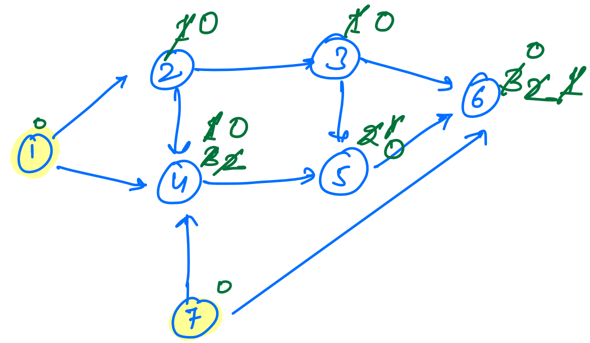
```
for( i = 1 ; i ≤ N ; i++){
        for( int nbr : Adj[i]){
                in[nbr] ++ ;
        }
}
```

if indegree[i] == 0
no pre-requisite

T.C → O(N+E)

Step-2. Insert all the nodes with indegree = 0 in a queue.



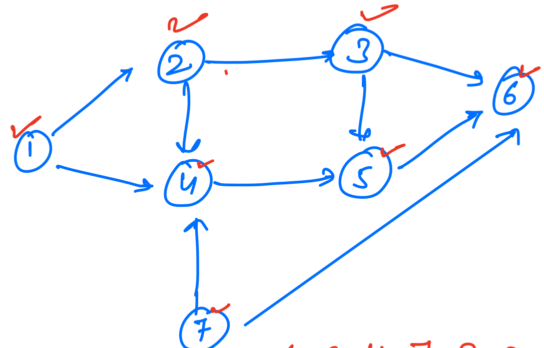


1, 7, 2, 3, 4, 5, 6 ✓

Step-3. Dequeue an element from queue & update the in-degree for all its neighbours. (decrement in-degree by 1)
If in-degree of any neighbour becomes 0, add that neighbour in the queue.

$$\boxed{\begin{aligned} T.C &\rightarrow O(N+E) \\ S.C &\rightarrow O(N) \end{aligned}}$$

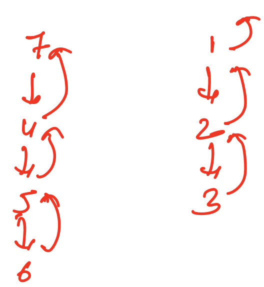
Right to Left.



∀i, visited[i] = false;

```
for( i = 1 ; i ≤ N; i++){
        if(visited[i] == false){
                dfs(i)
        }
}
```

6, 5, 4, 7, 3, 2, 1.



```
void  dfs( src){
    visited[src] = true
    for(nbr : Adj[src]){
        if( visited[nbr] == false){
            dfs(nbr)
        }
    }
    print (src)   // or insert it in stack for left to right order.
}
```

$$\boxed{\begin{array}{l} T.C \rightarrow O(N+E) \\ S.C \rightarrow O(N) \end{array}}$$

Disjoint Set Union

1
2
3

$s1$

4
8
6

$s2$

intersection

$s1 \cap s2 \rightarrow \{ \}, \rightarrow \phi \Rightarrow s1$ & $s2$ are disjoint sets.

$s1 \cup s2 \rightarrow \{1, 2, 3, 4, 8, 6\}$

union

Q. Given N elements. Consider each element as a unique set & perform multiple queries.
In each query check if (u, v) belongs to different sets,
if yes → merge the two sets & return true.
else → return false.

$N = 4$

1 2 3 4

Queries:

$(1,2) \rightarrow$ true.

$(3,4) \rightarrow$ true.

$(1,2) \rightarrow$ false.

$(1,4) \rightarrow$ true.

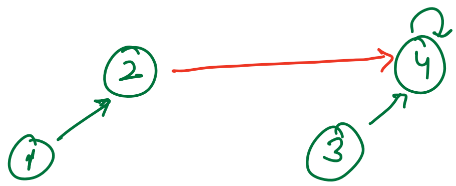
$(2,3) \rightarrow$ false.

idea → [Consider every set as a tree where every node points to parent and the root node points to itself.]

Queries:

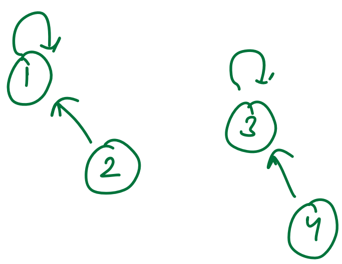
(1,2) → true
(3,4) → true
(1,2) → false.
(1,3) → true
(2,3) → false
(1,3) → false

parent[1] = 2 or parent[2] = 1
parent[3] = 4 or parent[4] = 3



parent[1] = 3, or parent[3] = 1 ✗

We can only update parent of root node.



Queries:

(1,2) → true
(3,4) → true
(1,2) → false.
(2,4) → true.

[Parent of root node is root node itself.]

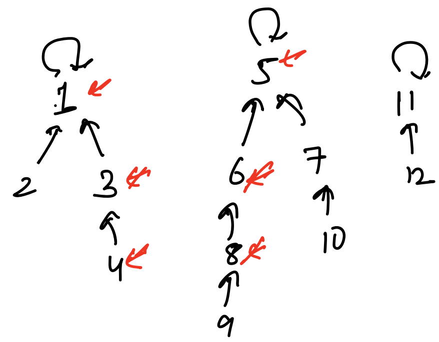
```
int root( int x) {
    while ( parent[x] != x) {
        x = parent[x]
    }
    return x;
}
```

O(N)

[T.C → O(Ht. of tree)]

```
boolean union ( int x, int y) {
    rx = root(x);
    ry = root(y);
    if(rx == ry) { return false }
    parent [rx] = ry        // parent[ry] = rx
    return true
}
```

[T.C → O(H) ~ O(N)]



(4, 8)
x   y

rx = 1

ry = 5

parent [ry] = rx

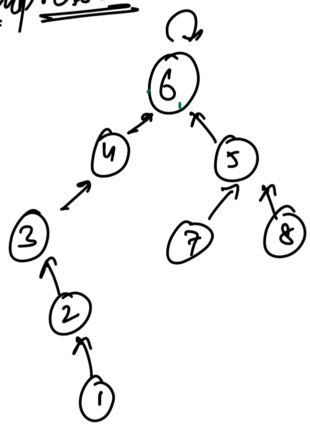
parent [5] = 1

Optimize D.S.U ?

→ Union by Rank ⇒ $O(\log_2 N)$ ⇒ (#todo)

→ Path Compression ⇒ $O(1)$ ✓

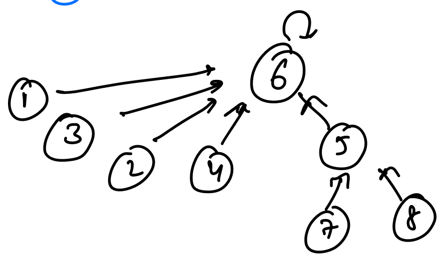
Path Compression



root(1)

✓ ✓ ✓ ✓

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6$



root(x) → K steps.

→ next time for these K elements ⇒ T.C → $O(1)$

```
int root( int x) {
    if (parent[x] == x ) { return x }
        r = root( parent[x]);
    parent[x] = r
    return r
}
```

root(6)) 6.

↑

root(4)) 6

↑

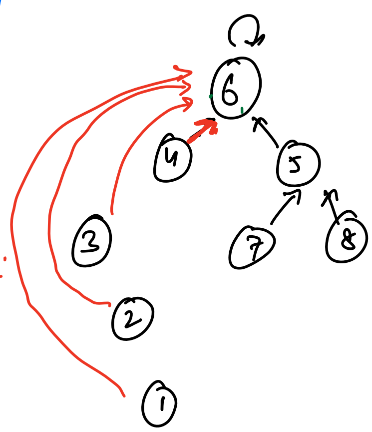
root(3)) 6

↑

root(2)) 6

↑

root(1)) 6.

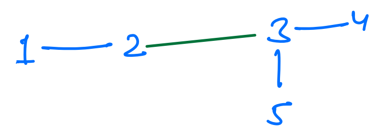


[T.C → $O(1)$ Amortized.

S.C → $O(N)$]

Applications Of D.S.U

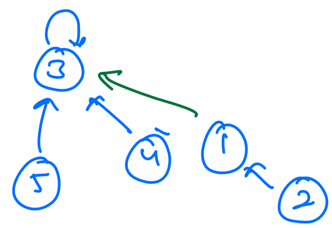
① check if graph is connected



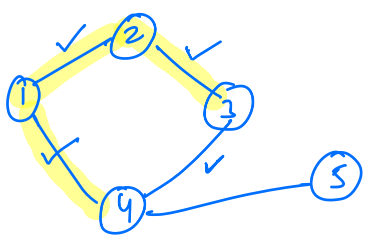
a) ∀ nodes, consider them as independent set.

b) ∀ edges, take union of (u,v)

c) If root is same for all the nodes then the graph is connected, otherwise not.



② Check for cycle in undirected graph →

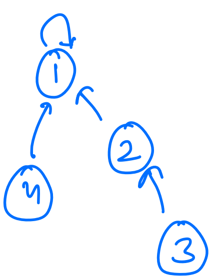


① ∀ nodes, consider them as independent sets.

② ∀ edges (u,v) → take union of (u,v)

if(union(u,v) == false) ⇒ cycle is present

otherwise ⇒ cycle is not present.



③ M.S.T. (Minimum Spanning Tree) --

to be continued