# Graphs 2


Good Evening

Thursday & Saturday

## Today's content

01. Topological sort

02. DSU (Disjoint set union)

Mock Interview / MBE

↳ unlock on the last day i.e on Saturday

↳ To give you guys the real Interview experience

& to unlock opportunities for you guys

↳ Total 12 mock interviews

4 mandatory + 8 floaters

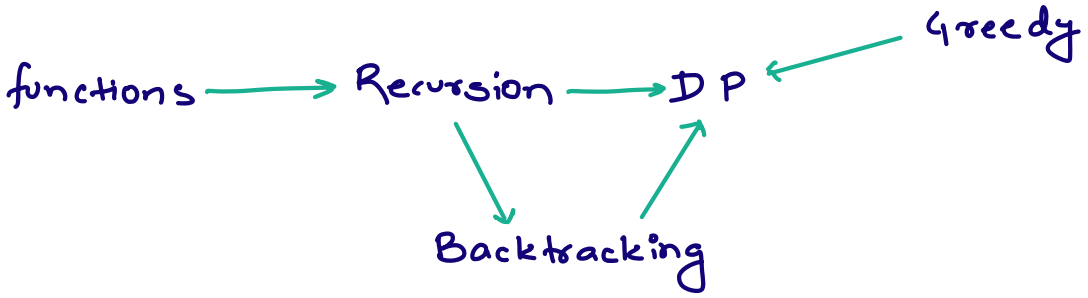
* 80% success rate when learners give mock interviews in between the module break.

⟶ Trust me, you will never feel prepared,

↳ Have this experience to understand where you're good at, where you lack. its not always about the knowledge, sometimes its about how you deliver it.

* → Preparation material → Advance DSA 1 & Advance DSA 2 } Whatsapp group

# Topological Sort




F G R B D ✓

F R G B D ✓

F R B G D ✓

F R B D G ✗

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

x is a prequisite of

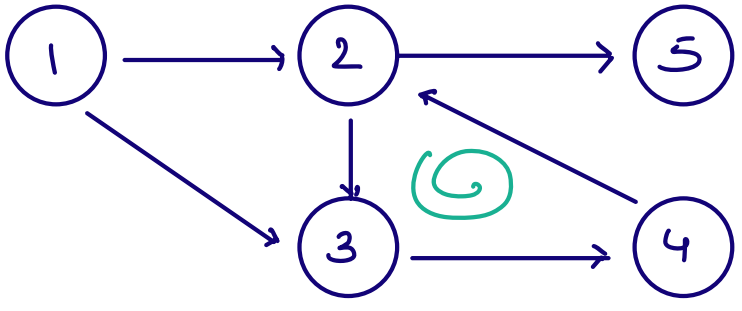
1 → 2, 3

2 → 5, 3

3 → 4

4 → 2

N = 5



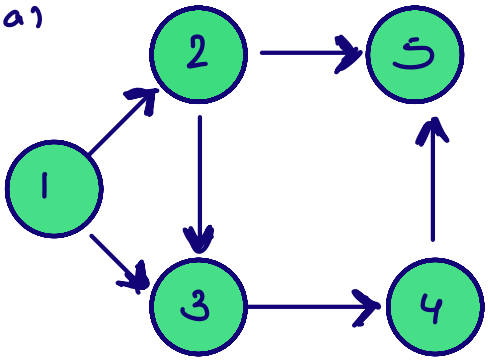
Solution → If there is a cyclic path, then we can't complete all the courses otherwise we can.

## Topological Sorting

Linear ordering of nodes such that if there is an edge from i → j, then i should be present be for j

Directed acyclic graph

(a)

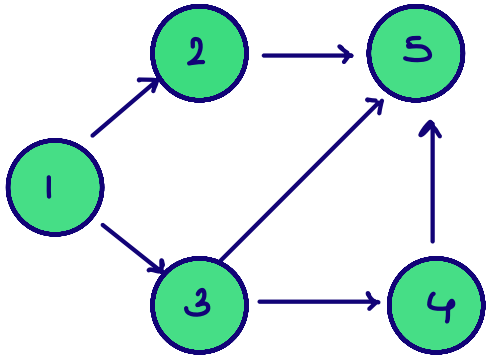



1 → 2 → 3 → 4 → 5

1 → 3 → 2 → 4 → 5

1 → 2 → 3 → 5 → 4

(b)

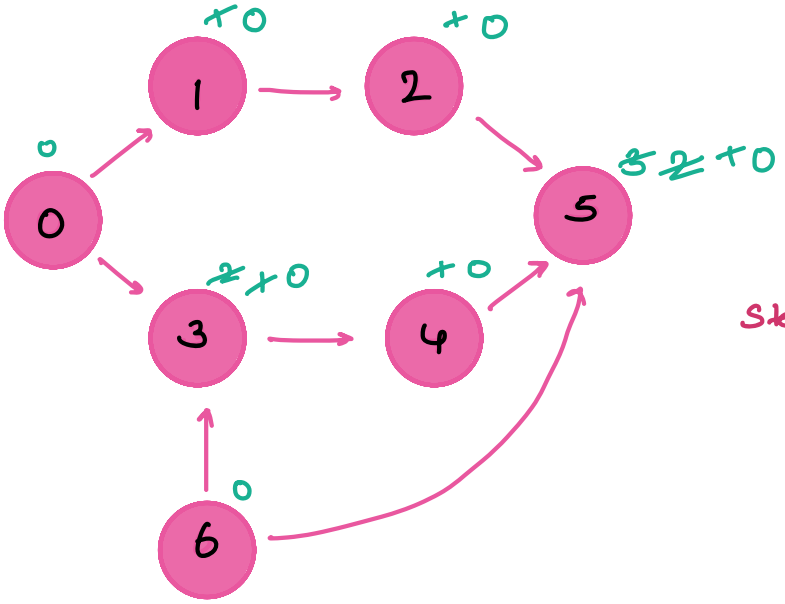



1 → 2 → 3 → 4 → 5

1 → 3 → 2 → 4 → 5

1 → 3 → 4 → 5 → 2

1 → 3 → 4 → 2 → 5




01 Left to Right (BFS)

Steps 1 - Find indegree of every node

```
indegree[N];

for(i=0; i<N; i++){
    for(int nbr : graph[i]){
        indegree[nbr]++;
    }
}
```

2 3

```
if (indegree[i]==0) No prerequisite
```

Step 2 → Insert ele in queue with indegree 0

Queue →

| ~~0~~ | ~~6~~ | ~~1~~ | ~~3~~ | ~~2~~ | ~~4~~ | ~~5~~ |
|---|---|---|---|---|---|---|

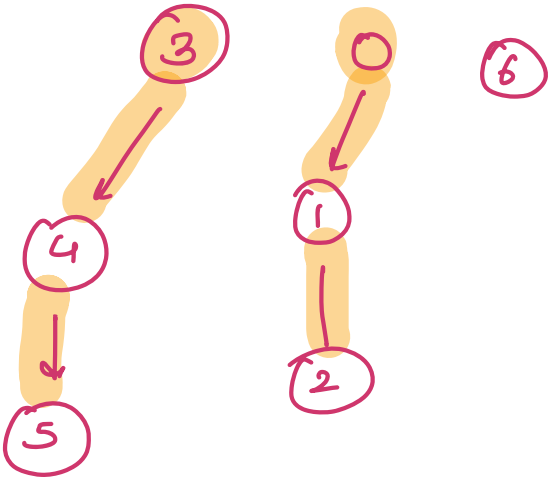
0 6 1 3 2 4 5 ← Ans

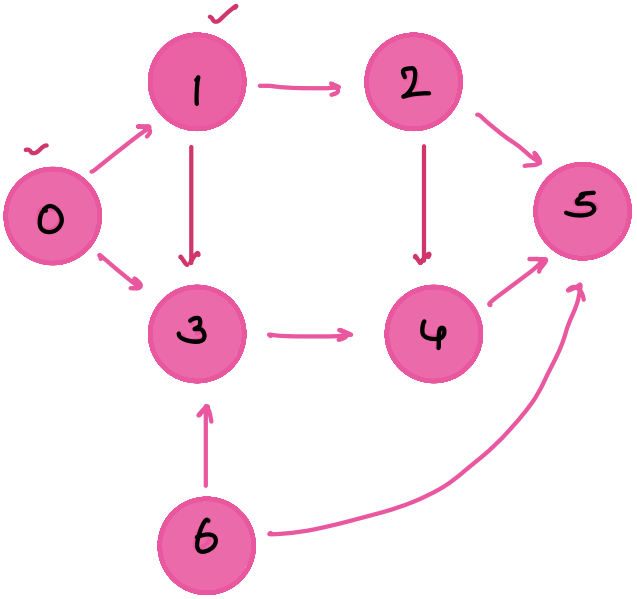
Step 3 → Deque an ele, print it & decrease the indegree of its neighbor by 1. If in[nbr]==0 Enqueue it inside the queue.

TC : O(N+E)

SC : O(N)

* Right to left (DFS)




6 0 1 2 3 4 5

```
visited[i] = false;

Stack<int> st;

for(i=0; i<N; i++){
    if(vis[i]==false){
        dfs(i, st, vis);
    }
}
```

TC : O(N+E)

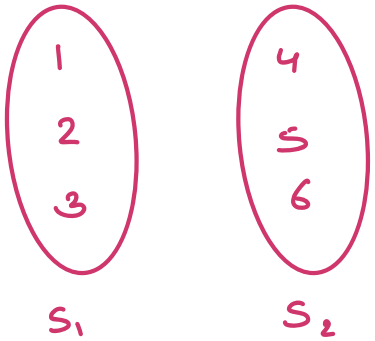
SC : O(N)

```
dfs (int src, int st, vis[])
    vis[src] = true ;
    for (int nbr : graph[src]) {
        if (vis[nbr] == false) {
            dfs (nbr, st, vis) ;
        }
    }
    st.push(src) ;
```

03. Empty the stack & print the elements

10:02 pm → 10:13 pm

* **Disjoint Set Union**




$S_1 \cup S_2 = \{1, 2, 3, 4, 5, 6\}$ (Union)

$S_1 \cap S_2 = \{\phi\} = \{\}$ (Intersection)

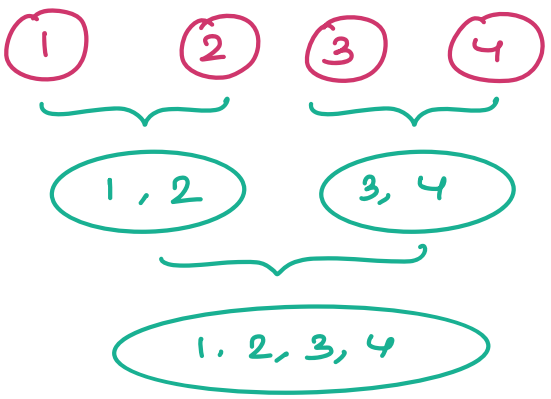
Q Given N elements. Consider each element as a unique set & perform multiple queries. In each query, check if (u, v) belongs to different sets

If yes → merge two sets & return true

else → return false.

Queries

N = 4




(1, 2) → true

(3, 4) → true

(1, 2) → false

(1, 3) → true

(2, 3) → false

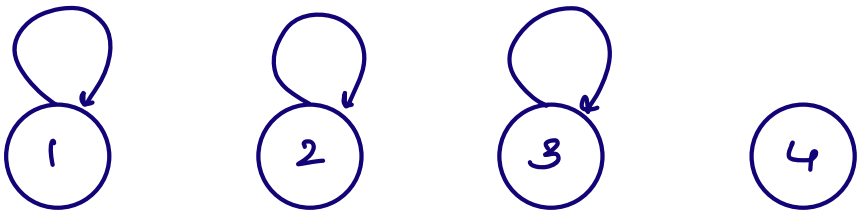
* Idea

01. Consider every set as a tree

02 Every node is going to point to its parent node

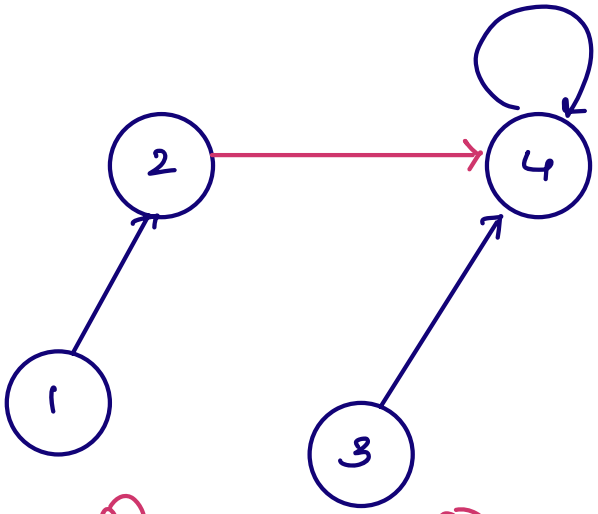
03. Root node point to itself

N = 4




par[1] = 2 or par[2] = 1

par[3] = 4 or par[4] = 3




par[1] = 3 or par[3] = 1

Queries

(1, 2) → true

(3, 4) → true

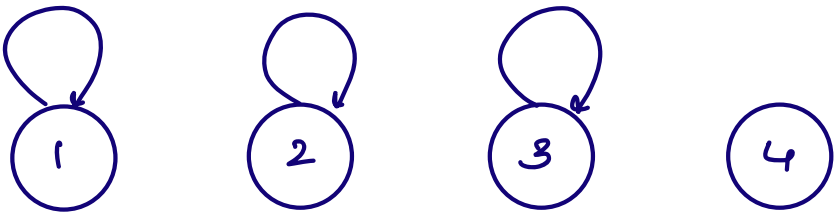
(1, 2) → false

(1, 3) → true

(2, 3)

Obs → If root node is same for u & v, then they belong to same set

Obs 2 → We should always parent of root node




## Queries

(1, 2) → true

(3, 4) → true

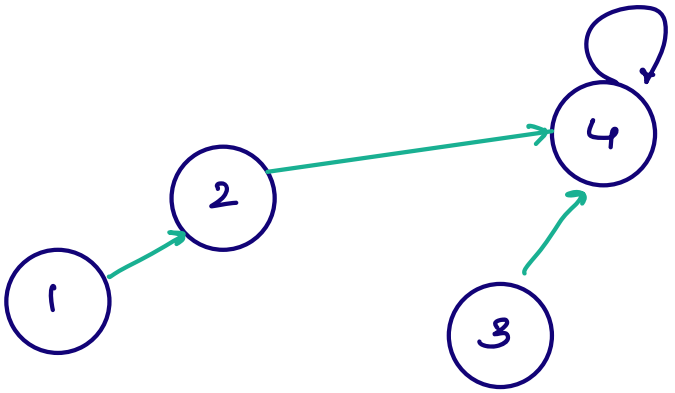
(1, 2) → false

(1, 3) → true

(1, 3) → false

par =

| 2 | 4 | 4 | 4 |
|---|---|---|---|
| 1 | 2 | 3 | 4 |




```
int root (int x)                                   TC: O(ht of tree)
    while (par[x] != x){
        x = par[x]
    }
    return x;
```

TC: O(Q * ht)

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

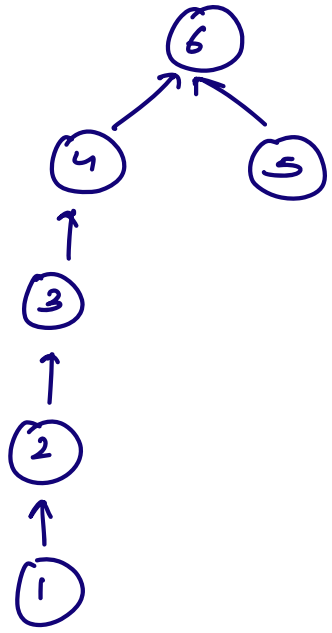
for one query

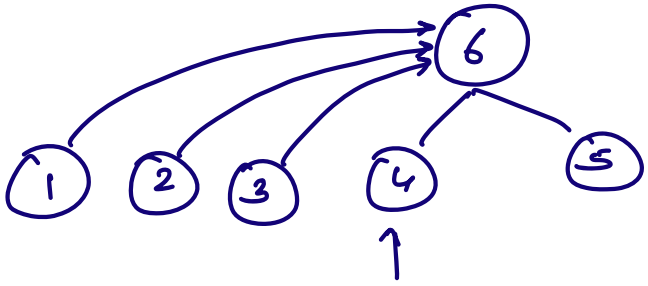
* **Optimise DSU**

01 Union by rank → TC : O(log n)

02. Path compression → TC : O(1)

Path compression



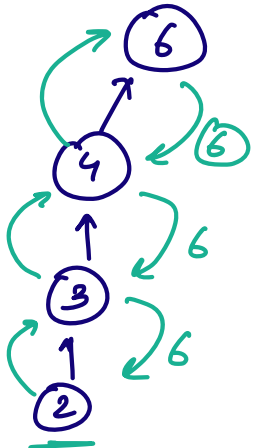




TC : O(1) Amortized TC

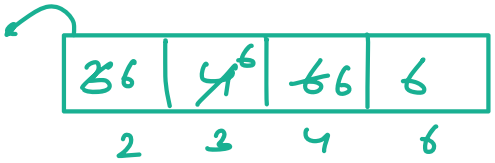
root (x) → k steps

- for next time optimise the TC to find root

```
int root (int x)
    if (par[x] == x) return x;
    r = root (par[x])
    par[x] = r;
    return r
```

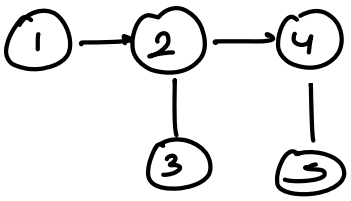


3



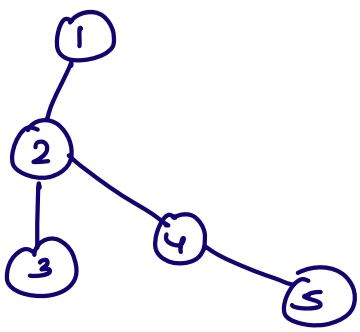

* Application of DSU

01. Check if graph is connected



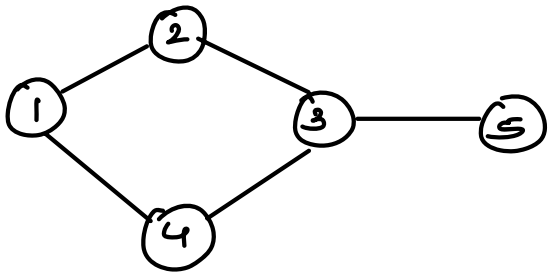

→ ∀ nodes, consider them as independent set

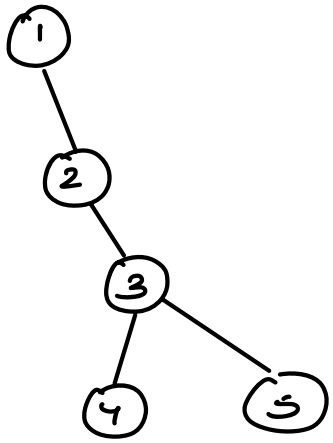
→ ∀ edges, take union of (u,v)




→ If all the nodes are having same root node, then entire graph is connected

02. Check for cycle in undirected graph




→ # nodes, consider them as independent set

→ # edges, take union of (u,v)

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

→ cycle is not present.