18|9|2023

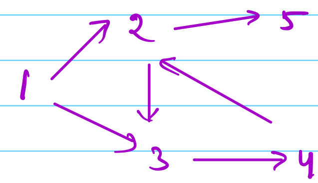
Graphs - 2

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

i/p : X is a prerequisite of . . . . . Adj list

1 → {2, 3}

2 → {3, 5}
3 → {4}
4 → {2}
5 → { }

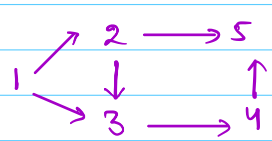


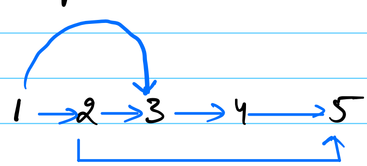
cyclic graph → false

acyclic graph → true.

Course schedule - I
Course schedule → II

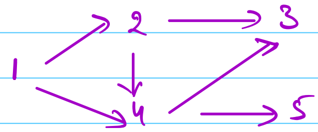
If it is possible to complete all the courses, find any one order to complete the courses.

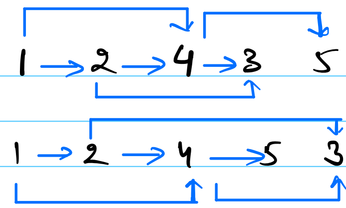




Directed Acyclic Graph (DAG)

Topological Sort → Linear ordering of nodes, s.t. if there is an edge from node i to j, then i will be on left of j

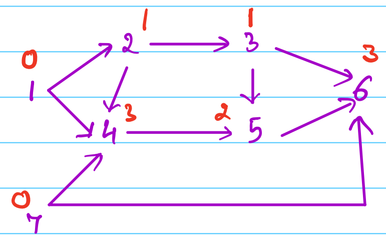




More than 1 topological order can be there for any DAG

Find topological order

1> **Left to right**



1> Compute indegree of nodes.

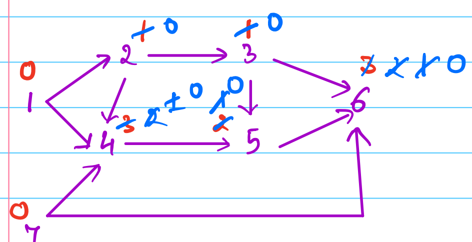
```
∀ i, in[i] = 0
for u -> 1 to N {
    for (v: adj[u]) {    u -> v
        in[v]++;
    }
}
```

TC: O(N+E)

2> Insert all nodes with indegree 0 in a set/ array.

3> Fetch any element from the set/ array, print it (ans) & update the indegree of adjacent nodes (decrease by 1)

4> If updated indegree of any node becomes 0, insert it in the set / array & repeat step 3 till all nodes are computed



set/ array.

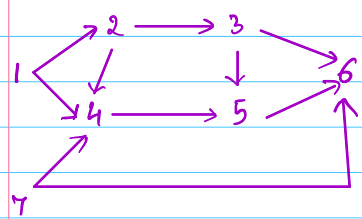
~~1~~, ~~7~~, ~~2~~, ~~3~~, ~~4~~ 5
6

TC: O(N+E)
SC: O(N)

o/p -> 1 7 2 3 4 5 6

```
2> for (u -> 1 to N) if (in[u] = 0) set.add(u)
3> curr = set.get() , for (v: adj[curr]) {
   print(curr)            ind[v]--;
                          if (ind[v] == 0) { set.add(v) }
                      }
```

2> **Right to left**



Topological sort can end if outdegree = 0 for a node.
↓
length of adjacency list

∀ i , vst[i] = false

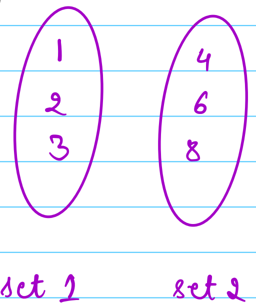
```
for i→ 1 to N {
    if (! vst[i] ) dfs(i)
}
```

TC: O(N+E)
SC: O (N)

```
void dfs (u) {
  vst[u] = true;
  for (v: adj[u]) {
     if (! vst[v]) dfs (v)
  }

  print (u); //right to left , for L to R store in stack.
}
```

Disjoint set Union (DSU)

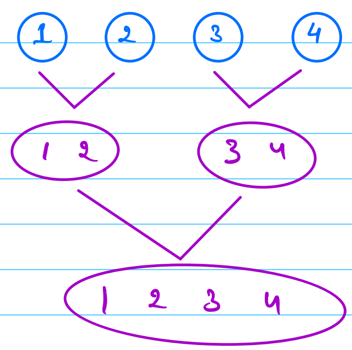


Union S1 ∪ S2 {1 2 3 4 5 6}

Intersection S1 ∩ S2 ϕ or { } — disjoint sets

Q Given N elements, consider each element a unique set & perform multiple queries. In each query, check if (U, V) belong to different sets, if yes → merge the 2 sets & return true, else return false.

Eg



Queries

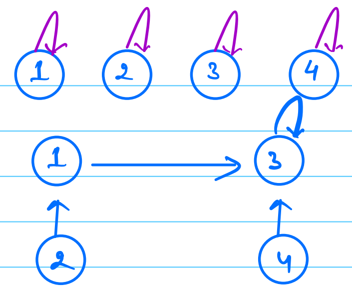
(1,2) → true
(3,4) → true
(1,2) → false
(1,4) → true
(2,3) → false

1) Consider every set as a tree.

2) ∀ nodes, node points to its parent.

3) ∵ for root there is no parent, root points to itself.

Eg



$$\begin{array}{cccccc} & 1 & 2 & 3 & 4 & \\ \text{parent} = [& \cancel{1} & \cancel{2} & 3 & \cancel{4} &] \\ & 3 & 1 & & 3 & \end{array}$$

(1,2) → T
(3,4) → T
(1,2) → F
(2,4) → T

(1,2) → parent[1] = 2 or
parent[2] = 1

(1,4) → parent[1] = 4 or ✓ It is only possible to update the parent of root node.
parent[4] = 1 ✗

How to find root for a given node.

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

TC: O(H) (height)

Check & union for query(u,v)?

```
boolean union (u,v) {
    x = root(u) , y = root(v)
    if (x == y) return false;
    parent[x] = y;
    return true;
}
```

TC: O(H+H) ≈ O(H)
O(N) → worst case.

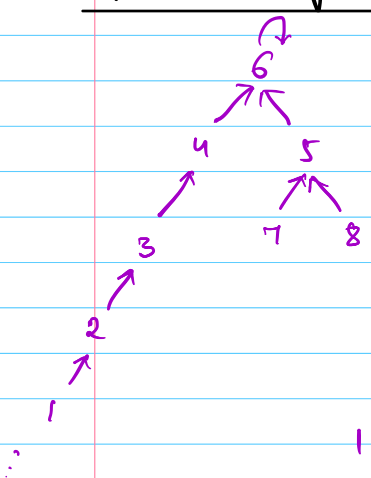
Meet at 8:38 am IST

Ways to optimize TC

1) Union by rank → HW. , TC: (log N)

2) Path compression → TC: O(K) for K nodes , TC: O(1)

Path compression

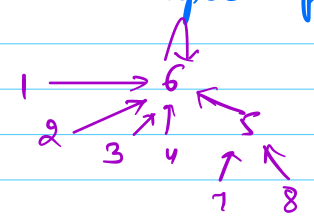


root(1) 1 → 2 → 3 → 4 → 6

we travelled 4 nodes before reaching root.

∵ ∀ 4 nodes , root is same

∴ update parent of each node to root.



Amortised TC: O(1)

	1	2	3	4	5	6
parent →	~~2~~ 6	~~3~~ 6	~~4~~ 6	~~5~~ 6	6	6

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

    parent[x] = root(parent[x])
    return parent[x];
}
```

Application of DSU

1> Check if the given graph is connected

Undirected graph → Travel complete graph from any node.

a> Consider every node as unique set.
b> ∀ edges (u,v) → take union (u,v)
c> If root ∀ nodes is same ⇒ connected

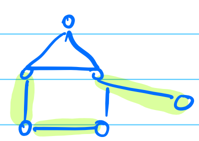
else ⇒ disconnected

TC: O(N+E)
SC: O(N)

2> Detecting cycle in an undirected graph

a> Consider every node as unique set.
b> ∀ edges (u,v) → take union (u,v)
if for any edge ⇒ union returns false

⇒ cycle is present.



Doubts