ALGORITHM ANALYSIS AND DESIGN

Module 2 Part 3

CST306

FRANKLIN'S LECTURES

KOSARAJU'S ALGORITHM

- It is a 2 Pass algorithm. Steps 1-4 are Pass1. Steps 5-7 are Pass2.

Pass 1

1. Set all vertices of graph G are unvisited.
2. Create an empty stack S.
3. Do DFS traversal on unvisited vertices and set it as visited. If a vertex has no unvisited neighbor, push it in to the stack.
4. Perform the above step until all vertices are visited
5. Reverse the graph G.
6. Set all nodes are unvisited.

$A \rightarrow B$

$A \leftarrow B$

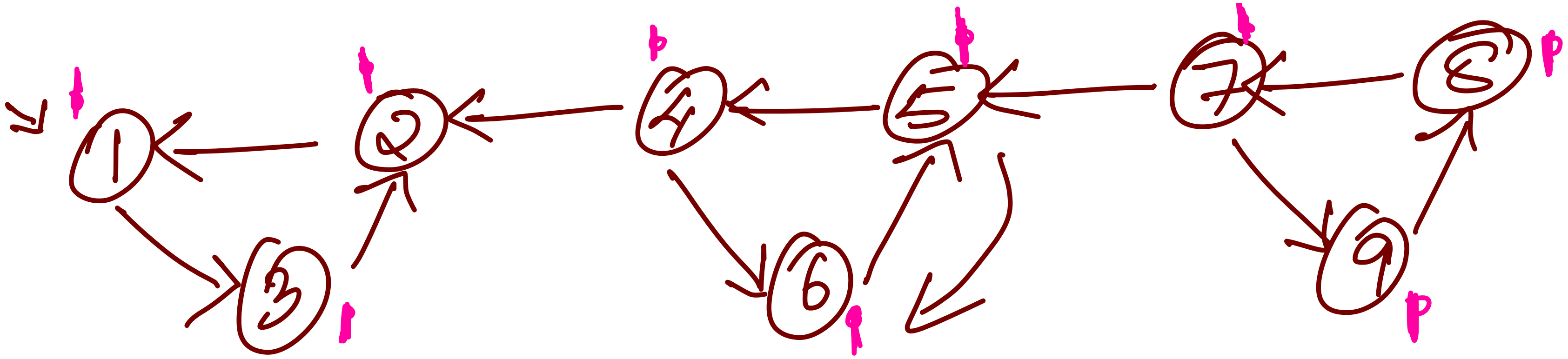
7. While S is not Empty

7.1 POP one vertex v’

7.2 If v’ is not visited

7.2.1 Set v’ as visited

7.2.2 Call DFS(v’). It will print strongly connected component of v’.



Pass 1 : 1.

2. Create an empty stack S

3. Perform DFS

stack.

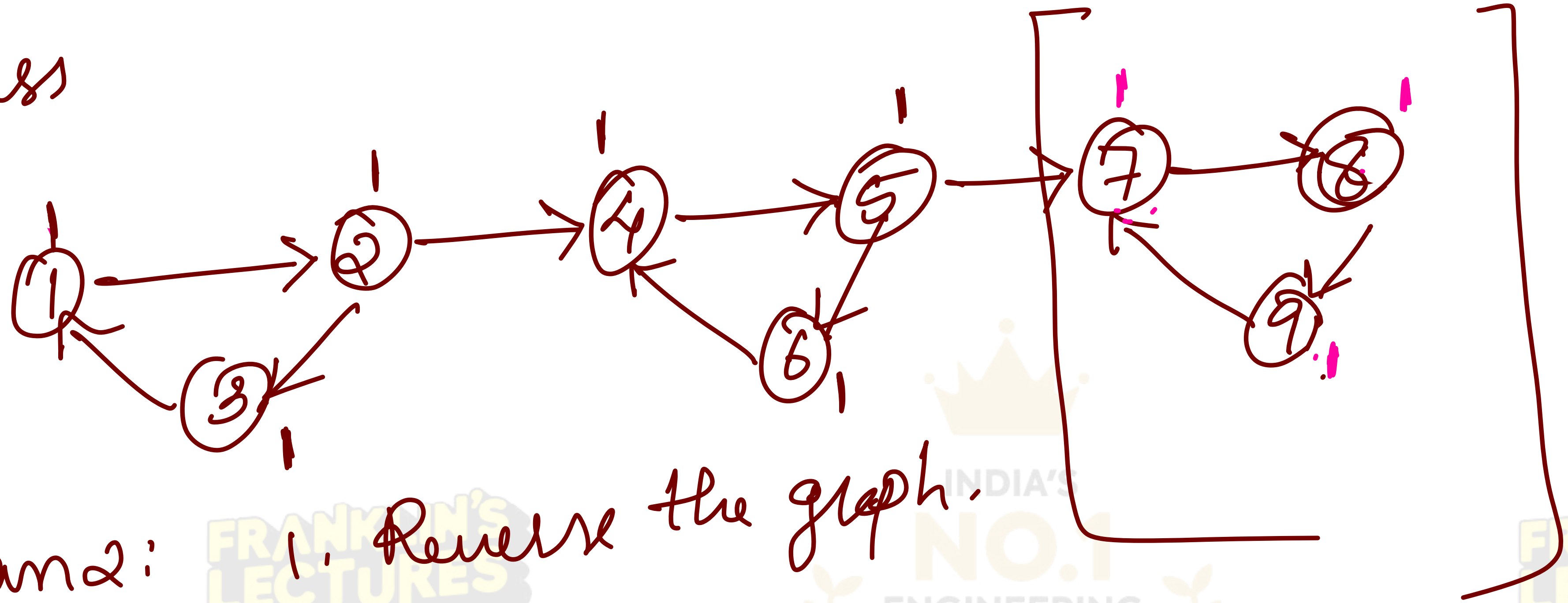
Pass 2 :

o/p SCC 1: 7 — 9 — 8

SCC 2: 4 — 6 — 5

SCC 2: 1 — 3 — 2

Pass



Pass2: 1. Reverse the graph.

TOPOLOGICAL SORTING

(1,2)

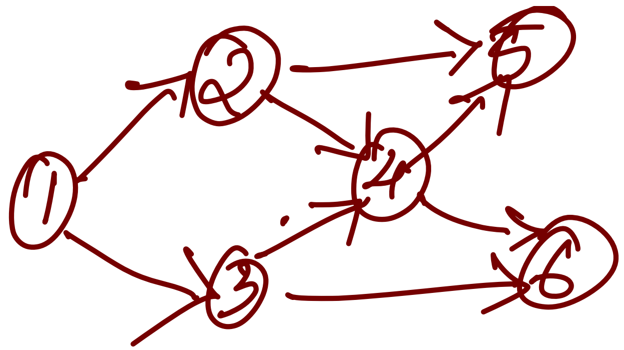
1 → 2

- Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge (u,v), vertex u comes before v in the ordering.
- A topological sort of a graph is an ordering of its vertices along a horizontal line so that all directed edges go from left to right.
- If the graph contains a cycle, then no linear ordering is possible.
- Topological Sorting for a graph is not possible if the graph is not a DAG.

ALGORITHM

1. Identify a node with no incoming edges(indegree=0)
2. Add this node to the ordering.
3. Remove this node and all its outgoing edges from the graph.
4. Repeat step 1 to 3 until the graph becomes empty.

Ex:



ts : 1, 2, 3, 4, 5, 6

1, 2, 3, 4, 6, 5

1, 3, 2, 4, 5, 6

1 3, 2, 4, 6, 5

Indegree

1 — 0
2 — 1
3 — 1
4 — 2
5 — 2
6 — 2

2 - 0
3 - 0
4 - 2
5 - 2
6 - 2.

3 - 0
4 — 1
5 — 1
6 — 2

4 - 0
5 - 1
6 - 1

5 - 0
6 - 0

2 - 0
4 — 1
5 - 2
6 - 2.

4 - 0
5 - 1
6 - 1

5 - 0
6 - 0

DISJOINT SETS

- Two or more sets with nothing in common are called disjoint sets.
- Example : $S1 = \{1, 2, 3, 4\}$ $S2 = \{5, 6, 7\}$ $S3 = \{8, 9\}$
- Two sets S1 and S2 are said to be disjoint if $\boxed{S1 \cap S2 = \phi}$
- The disjoint set data structure is also known as union-find data structure and merge-find set.

Disjoint set representations

- Linked list representation(Tree representation)
- Array representation

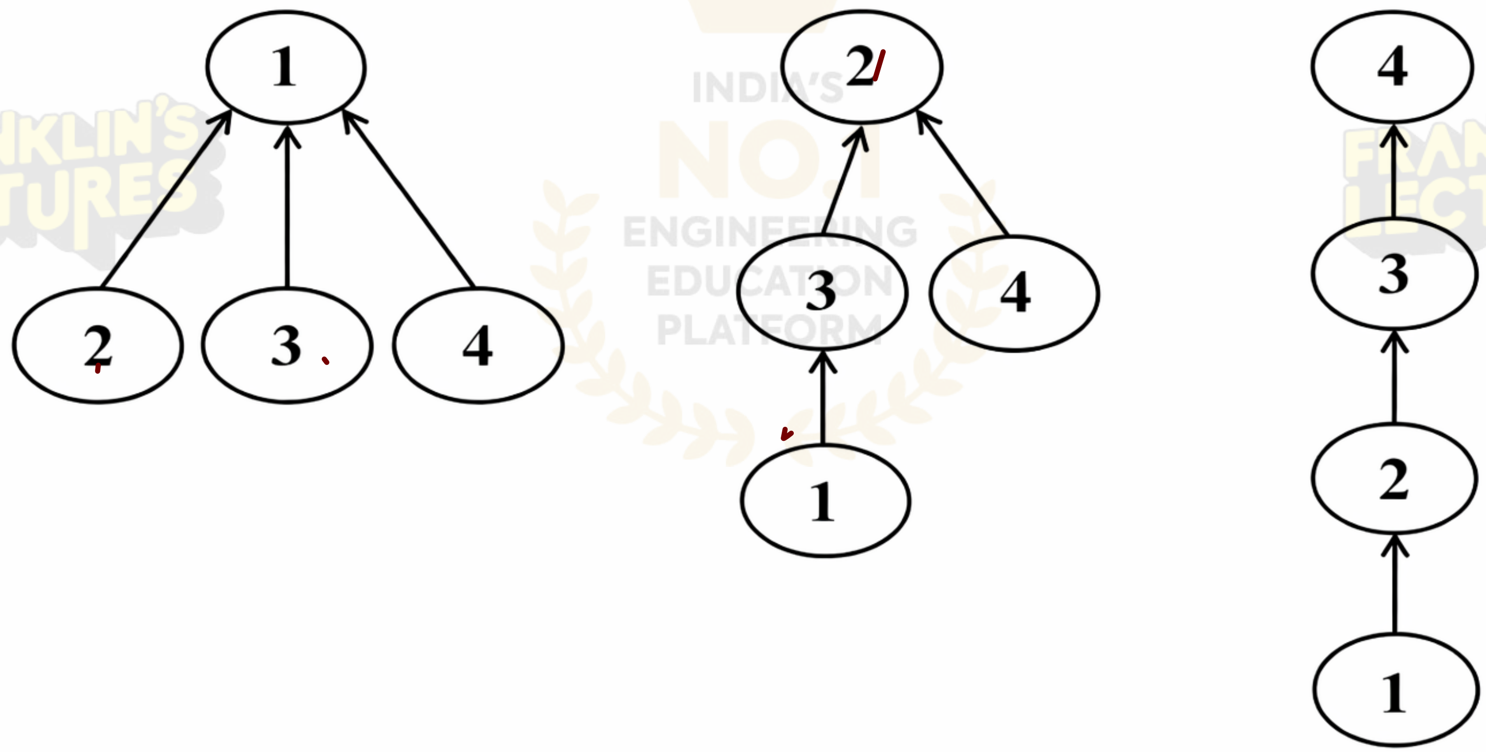
$S_2 = \{30, 40\}$

$S_1 = \{10, 20\}$

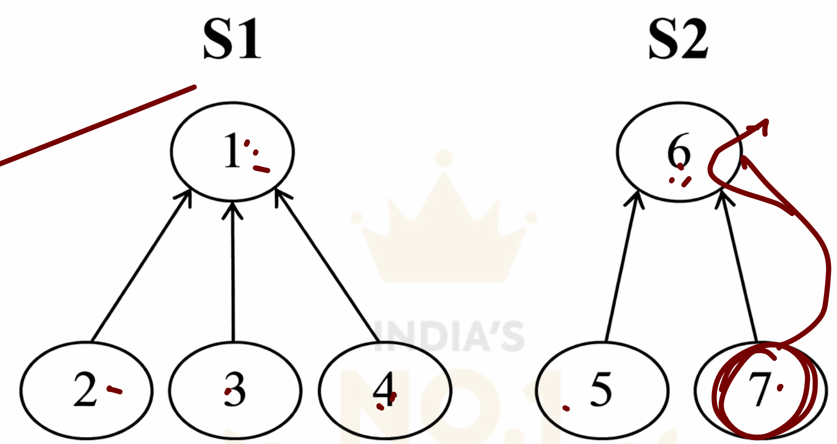
Linked list representation(Tree representation)

Example: S1 = {1, 2, 3, 4} — node.

This set can be represented as tree in different ways



- Example: S1 = {1, 2, 3, 4} S2 = {5, 6, 7}



- Array representation
- Example: S1 = {1, 2, 3, 4} S2 = {5, 6, 7}

i	1	2	3	4	5	6	7
p	-1	1	1	1	6	-1	6

Parent

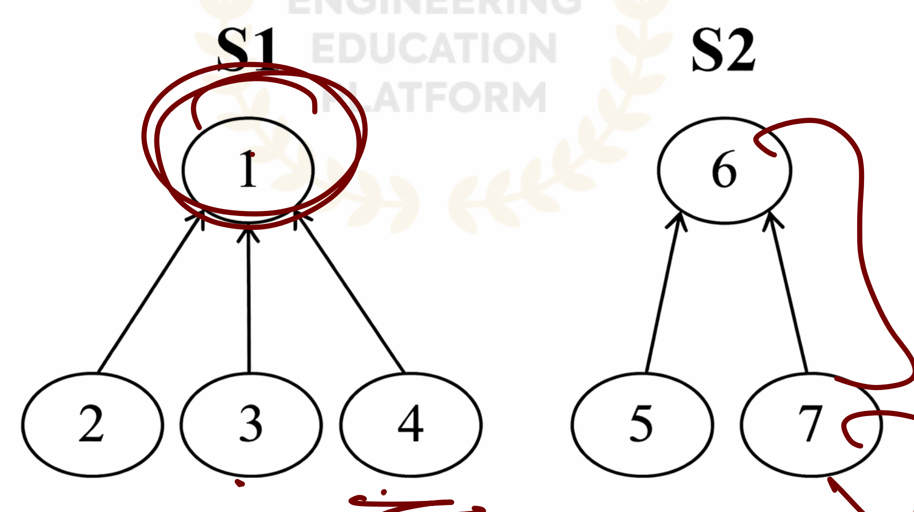
DISJOINT SET OPERATIONS

- Make set
- Union
- Find
- **Make Set operation**
- Make-set(x) function will create a set that containing one element x.
- Algorithm Make-set(x)

 1. Create a new linked list that contains one node n
 2. n   data=x
 3. n   parent = NULL

Find Operation

- Determine which subset a particular element is in.
- This will return the representative(root) of the set that the element belongs.
- This can be used for determining if two elements are in the same subset.



- Find(3) will return 1, which is the root of the tree that 3 belongs
- Find(6) will return 6, which is the root of the tree that 6 belongs

Find Algorithm

Algorithm Find(n)

1. while n parent != NULL do

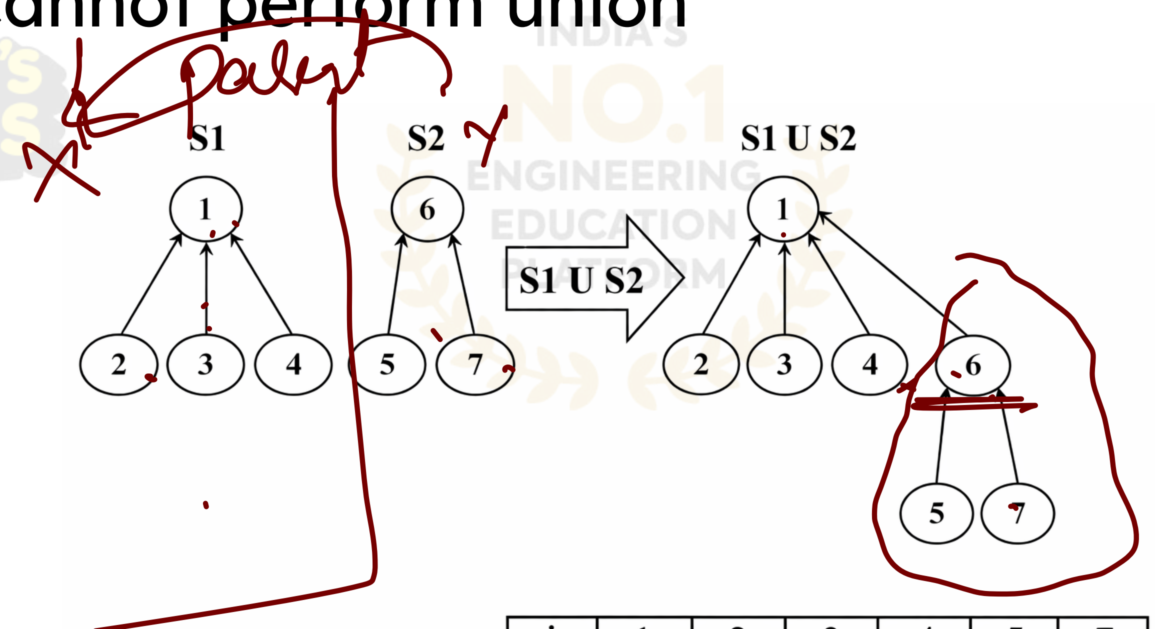
    1.1 n = n parent

2. return n

Worst case Time Complexity = O(d), where d is the depth of the tree

UNION OPERATION

- Join two subsets into a single subset.
- Here first we have to check if the two subsets belong to same set. If no, then we cannot perform union



X ≠ Y

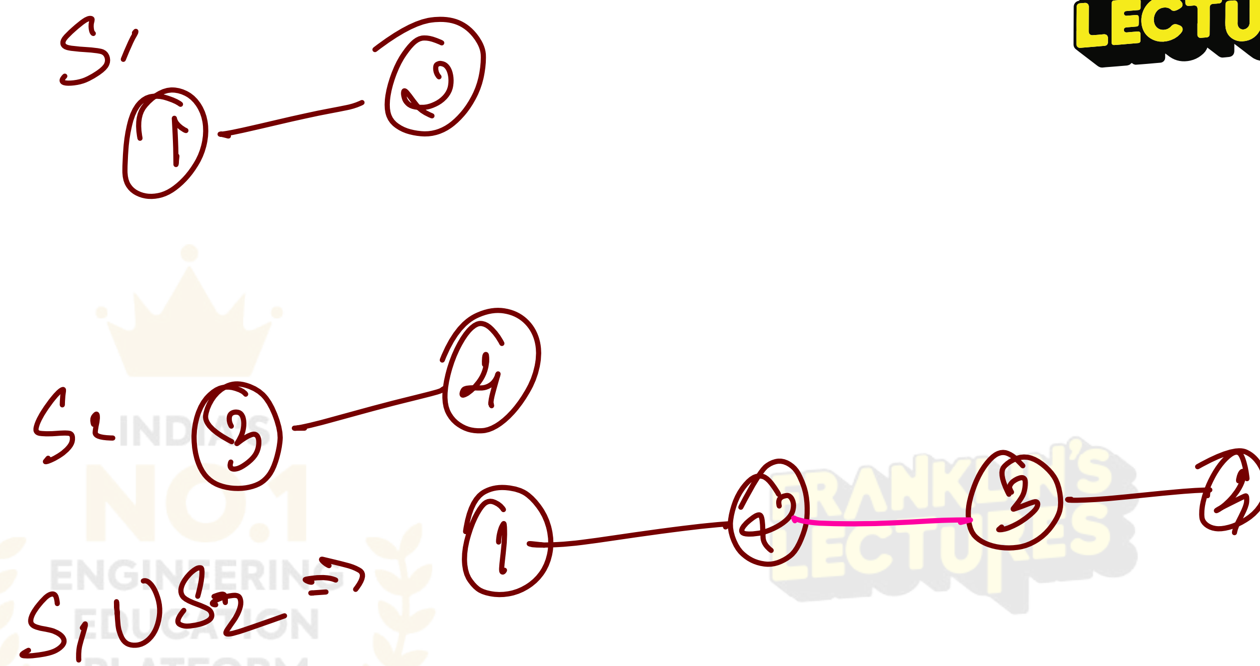
X ← Parent of Y

i	1	2	3	4	5	7
P	-1	1	1	1	6	6

UNION ALGORITHM

Algorithm Union(a, b)

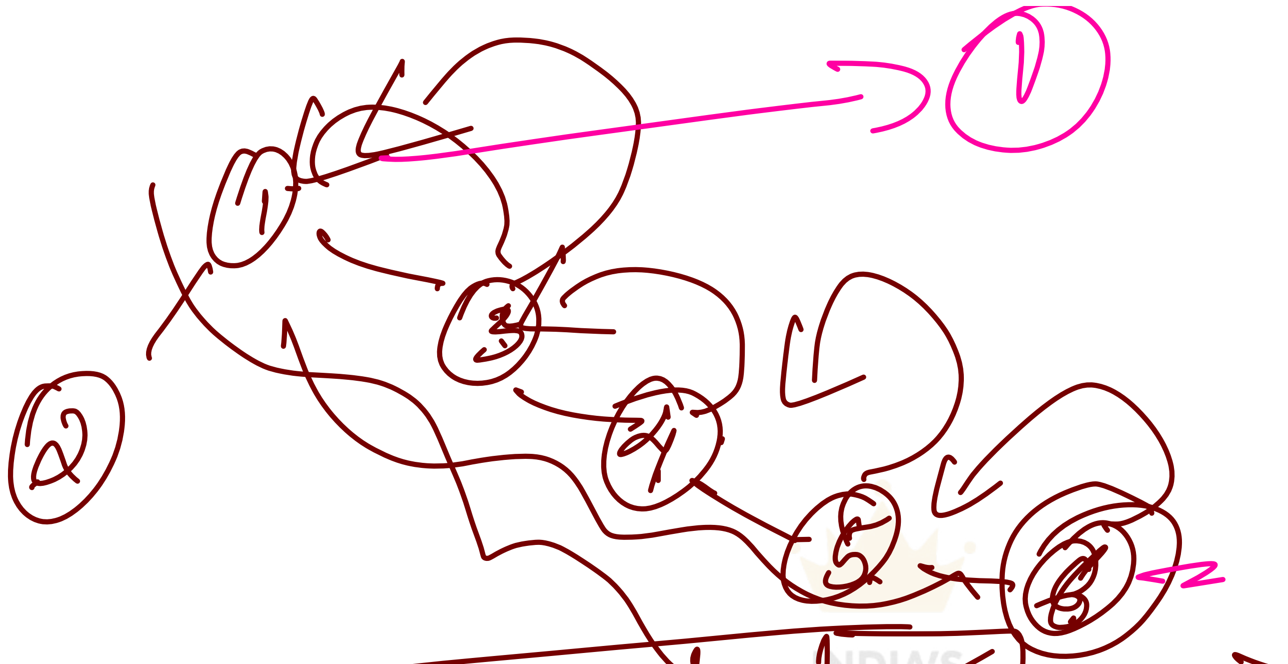
1. X =Find(a)
2. Y = Find(b)
3. If X != Y then
    1. Y parent = X



- Worst case Time Complexity = O(d), where d is the depth of the tree.

- There are two ways to improve the time complexity of Find and Union operation:

1. Path Compression
2. Union by Rank



P[i] =

1	2	3	4
4	1	1	3

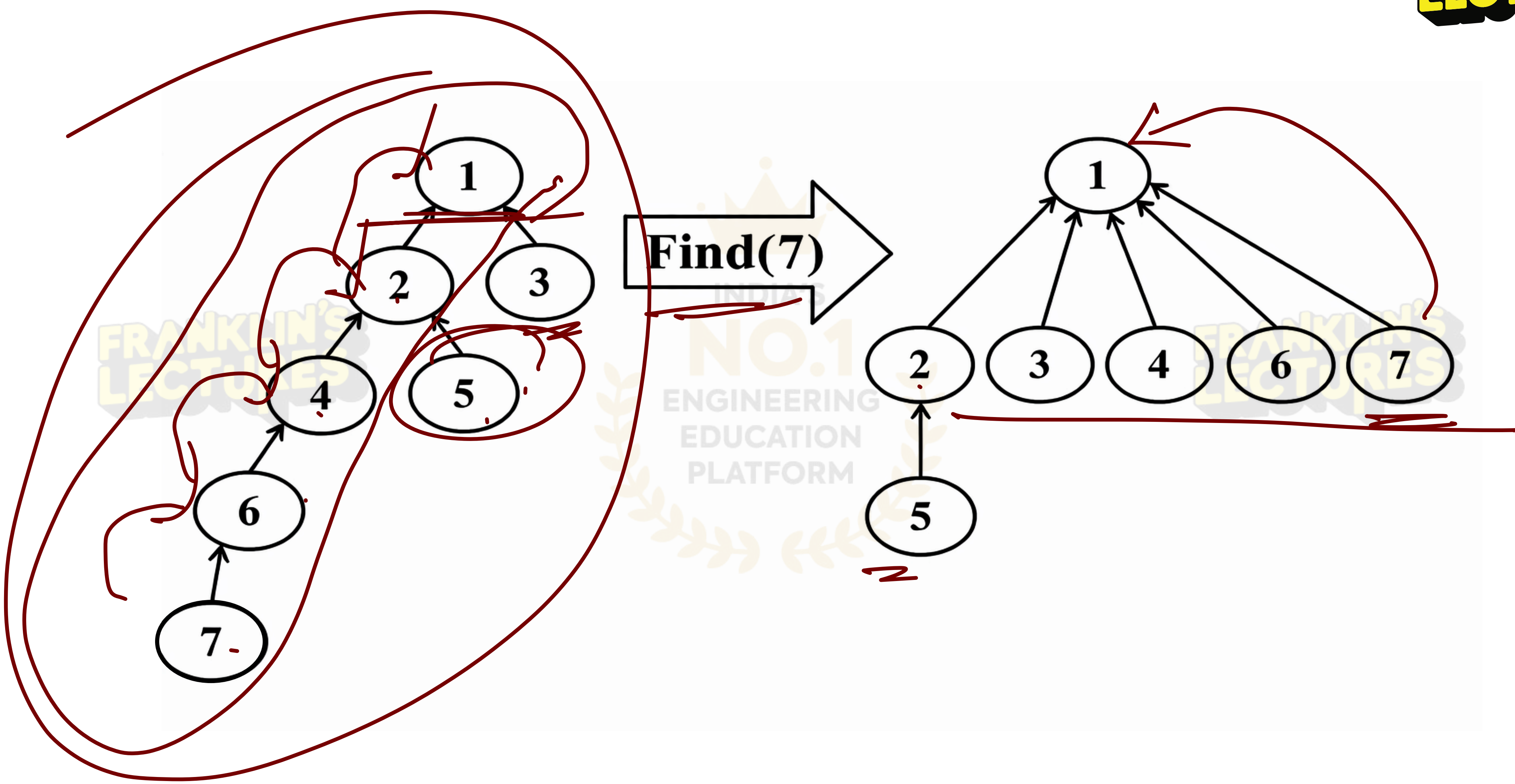
→ . - - - .

Path Compression(collapsing rule)

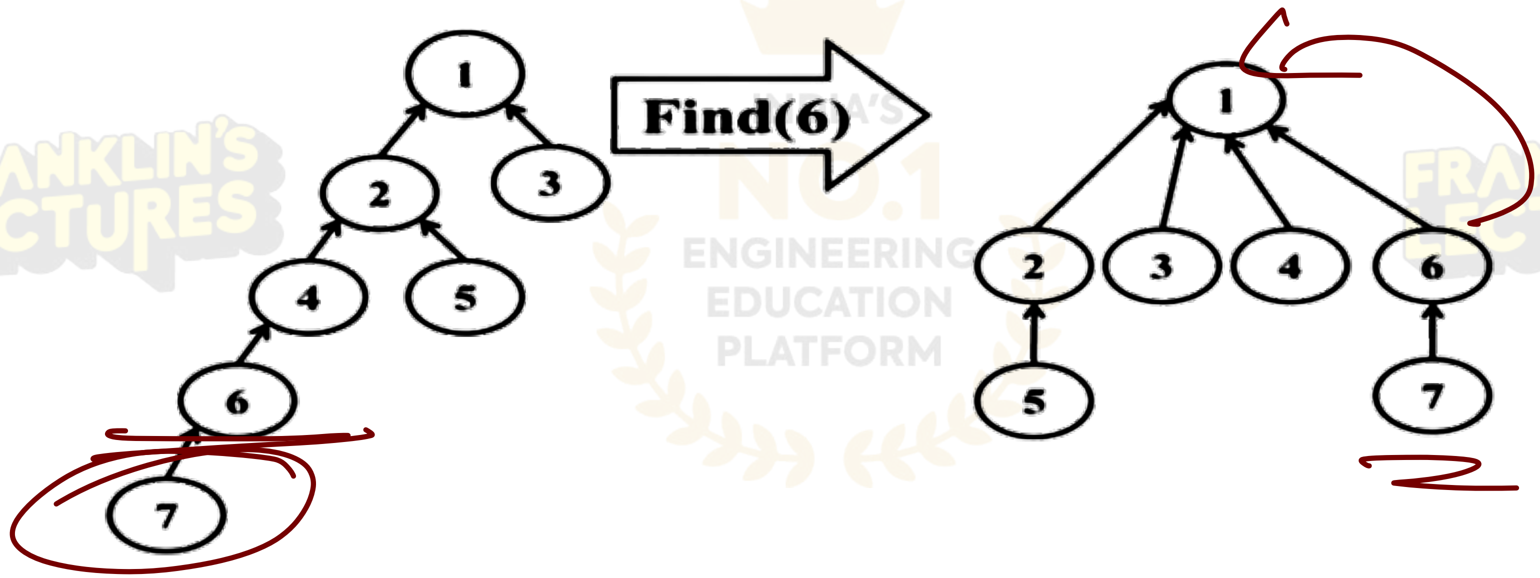
- Path compression is a way of flattening the structure of the tree whenever Find is used on it.
- Since each element visited on the way to a root is part of the same set, all of these visited elements can be reattached directly to the root.

- Example

Find (7) = 1



- Next time we perform Find(6) it will give us the answer in two steps instead of four prior to the optimisation.
- Example:

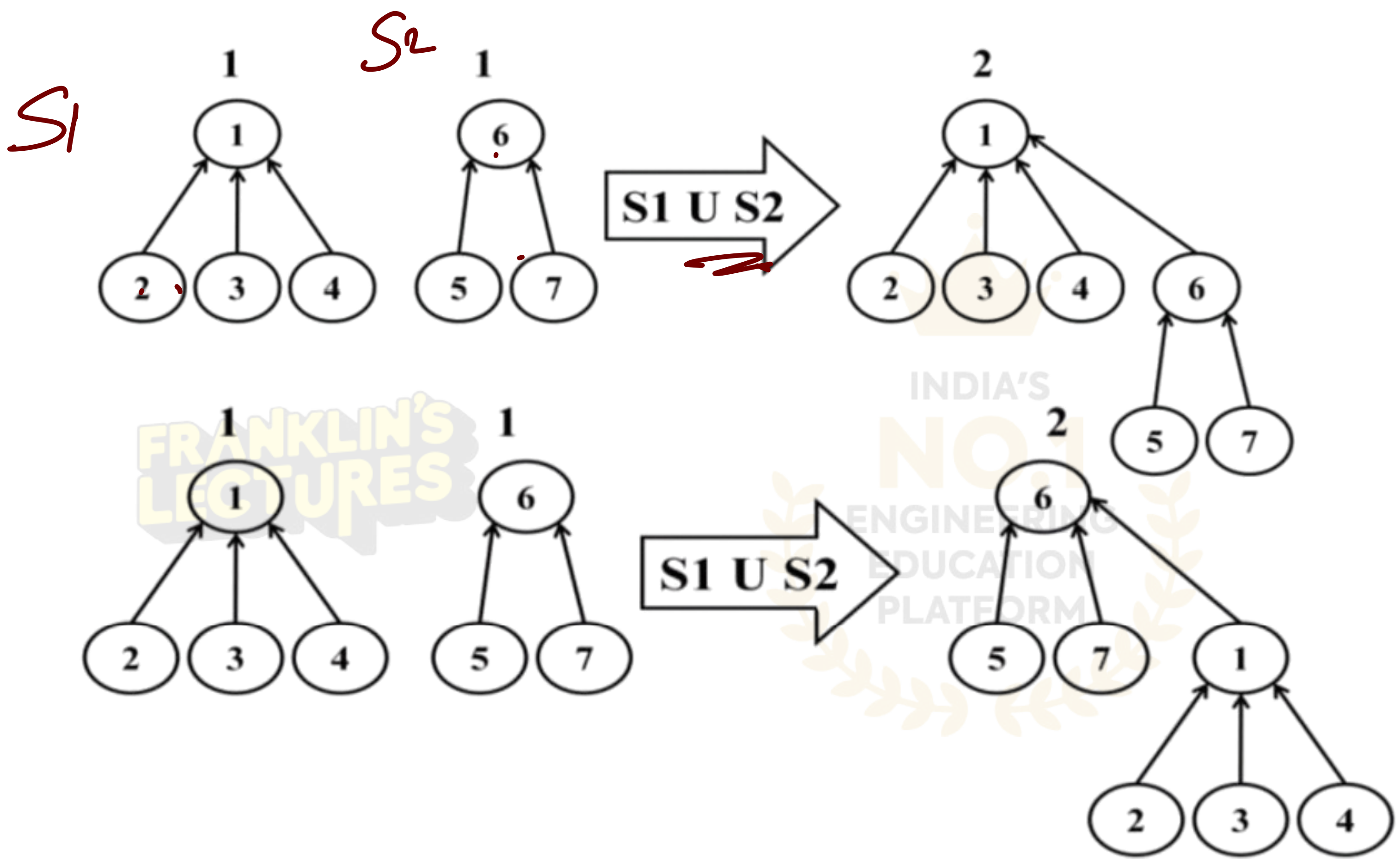


UNION BY RANK(WEIGHTED RULE)

S_1 S_2

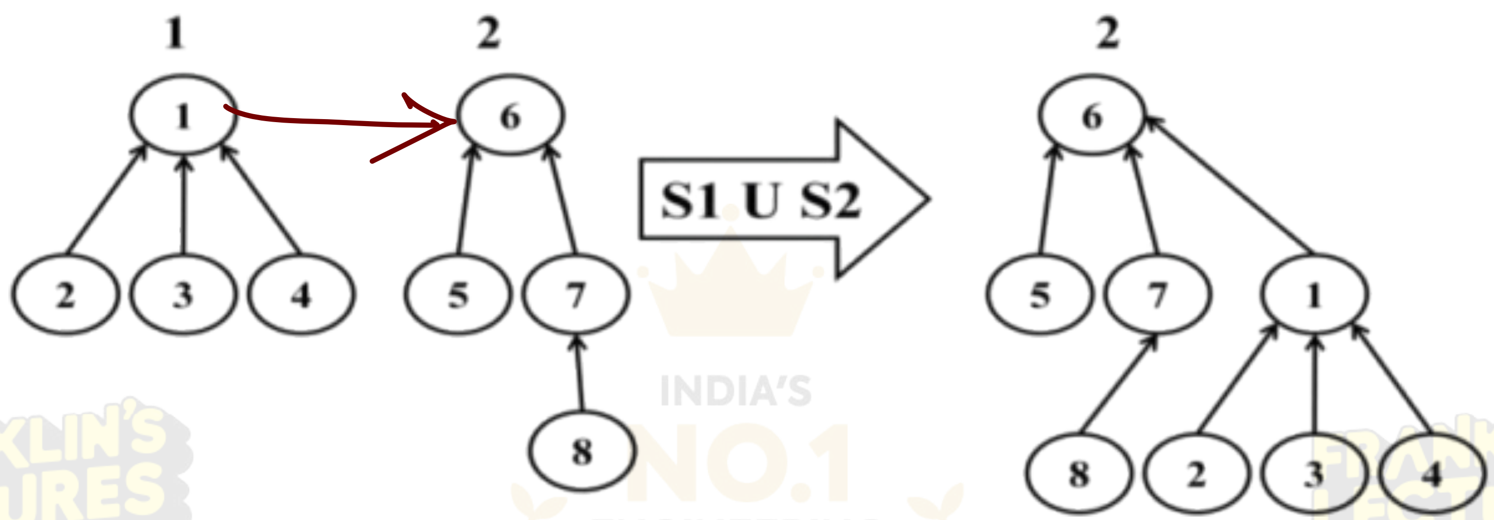
- In this operation we decide which tree gets attached to which.
- The basic idea is to keep the depth of the tree as small as possible.
- We assign a new value Rank to each set. This rank represents the depth of the tree that the given set represents.
- If we union two sets and :
  - Both sets have same rank ⟶ then the resulting set's rank is 1 larger.
  - Both sets have the different ranks — the resulting set's rank is the larger of the two.

• Example



we can union these two sets in the above two ways

- Example



Applications of disjoint sets

- It is easier to check whether the given two elements are belongs to the same set.
- Used to merge 2 sets into one

THANK YOU

FRANKLIN'S LECTURES