

# The Graph Algorithms Course

## Pre-Requisite

- Basic Programming (we will code in C++)
- Basic - Intermediate Recursion
- Arrays, HM, LL, Trees

## Outcome Expectation

- Course will be starting from very basics  
but we will cover adv cf topics
- Beneficial with Internen perspective too.

## Topics

(21 classes)

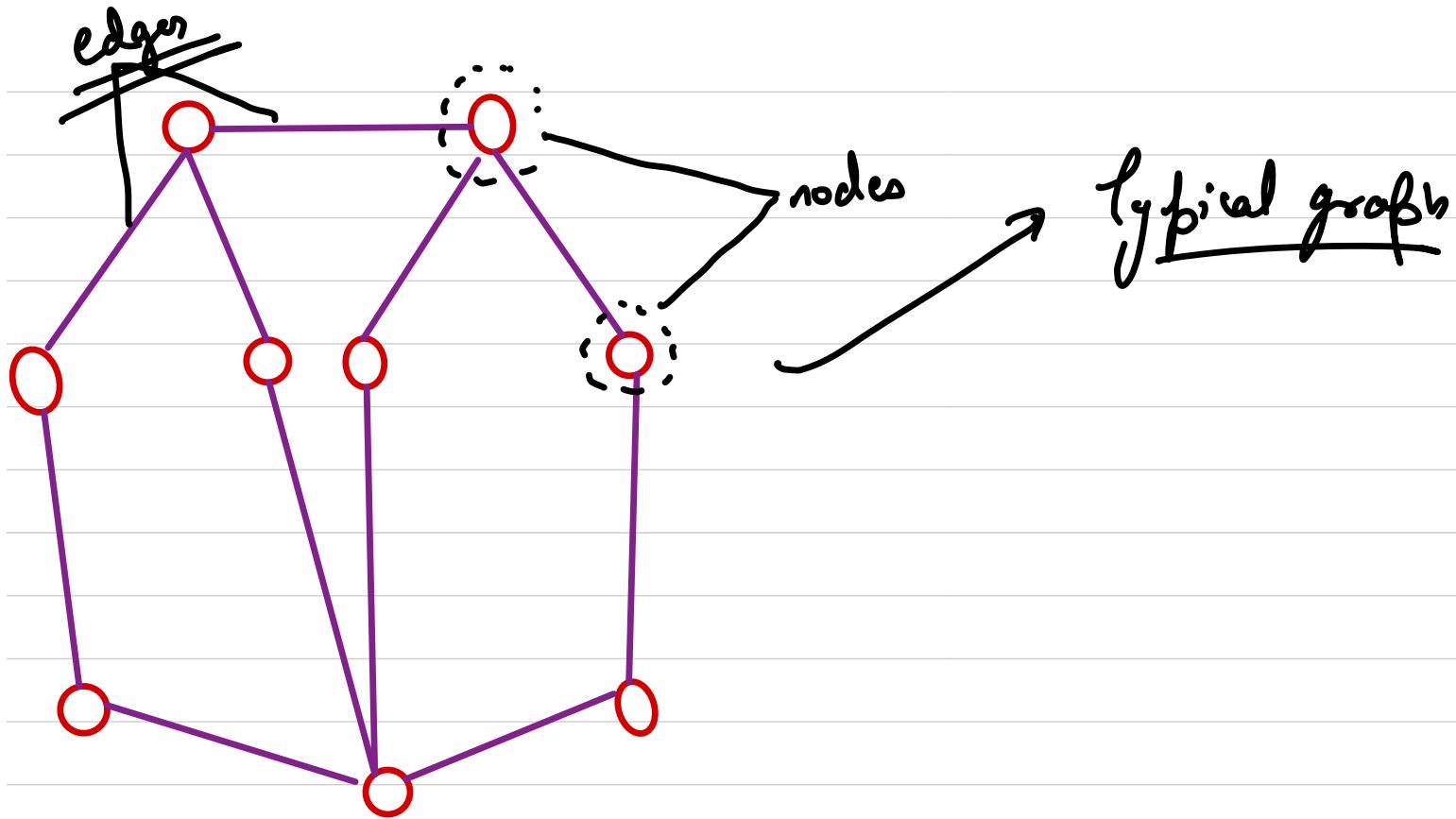
- Basic graphs & terminologies
- Traversal techniques      Bfs / Dfs
- DP on graph (DP on Adj)
- Matrix Bfs / bfs , 0-1 Bfs dc
- Trees (n-ary) , DP on trees
- Shortest path, MST
- DSU
- SCC , Bridges / articulation point

## Graphs



What is a graph ??

A graph is a collection of nodes and edges where each node might point to / connect to other nodes. The nodes represent real life entities and are connected by edges representing relationship between the nodes.



# # Applications of graph

Biology → PPI graph  
→ Metabolic network graph

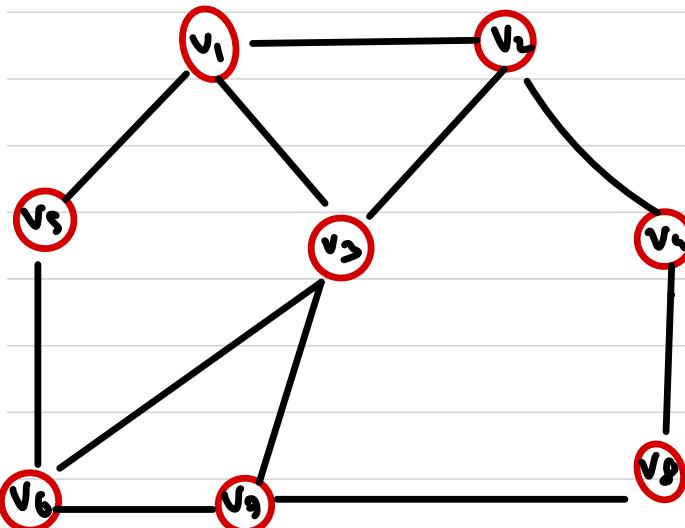
Electrical → Circuit net graph

Computer  
Science → Shortest path (map)  
→ gsm frequency assignment  
→ routing mechanism  
→ social media

## formal definition of graph

set of vertices  
 $V = \{v_1, v_2, v_3, \dots, v_n\}$

~~Edge set~~



$E = \{ \{v_1, v_2\}, \{v_2, v_3\}, \dots \}$  → unordered set

$$G = (V, E)$$

→ mathematical notation of graph

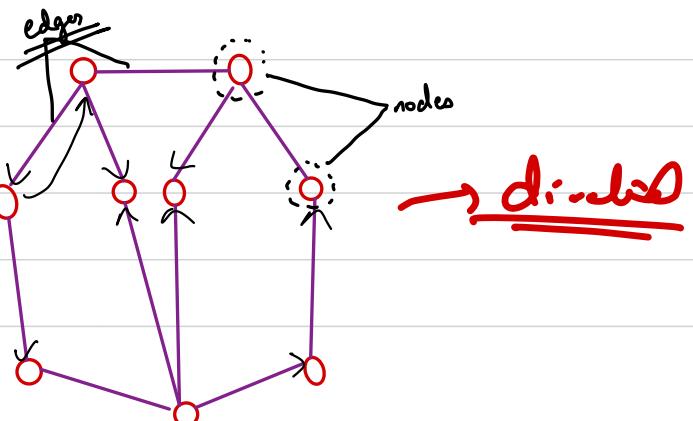
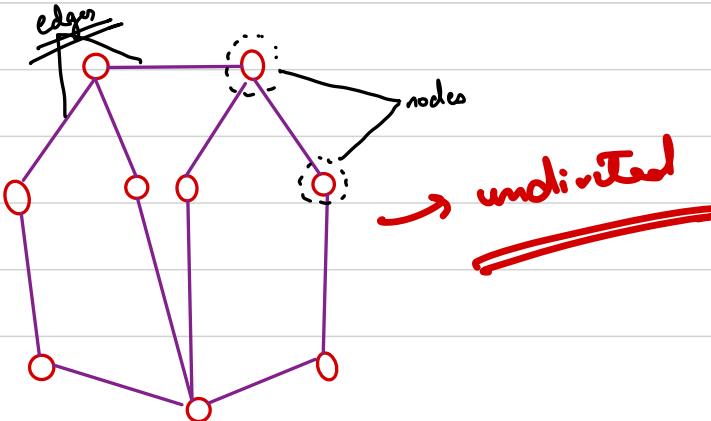
/unordered

A graph  $G$  is an ordered pair of a set  $V$  vertices &  $E$ , edges.

# Types of graph

# Based On direction

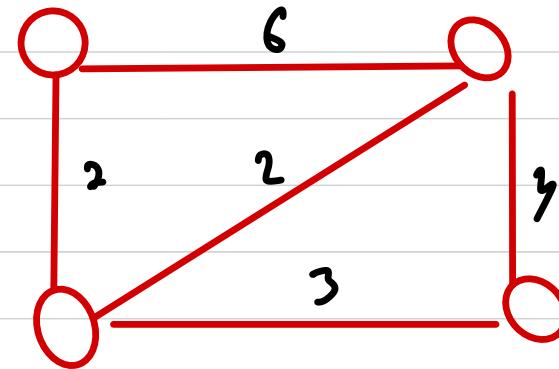
- **Directed** → water flow , road network,
- **Undirected** → facebook



# Based on edge property

① Weighted

② Unweighted



# Based on edge density

1) Sparse

2) Dense

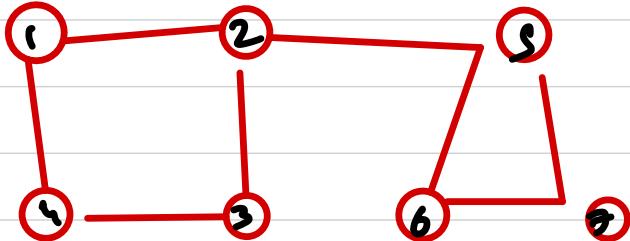
# # Representing graphs as a data structure

1) Adjacency Matrix

2) Adjacency List → adjacency set/map  
3) Incidence Matrix ↗ edge list

# # adjacency matrix

$A_{ij} = \begin{cases} 1 & \rightarrow \text{if there is an edge from } i \text{ to } j \\ 0 & \rightarrow \text{else} \end{cases}$



	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	1	0	1	0	1	0	0
3	0	1	0	1	0	0	0
4	1	0	1	0	1	0	0
5	0	1	0	1	0	1	1
6	0	0	0	0	1	0	1
7	0	0	0	0	1	1	0

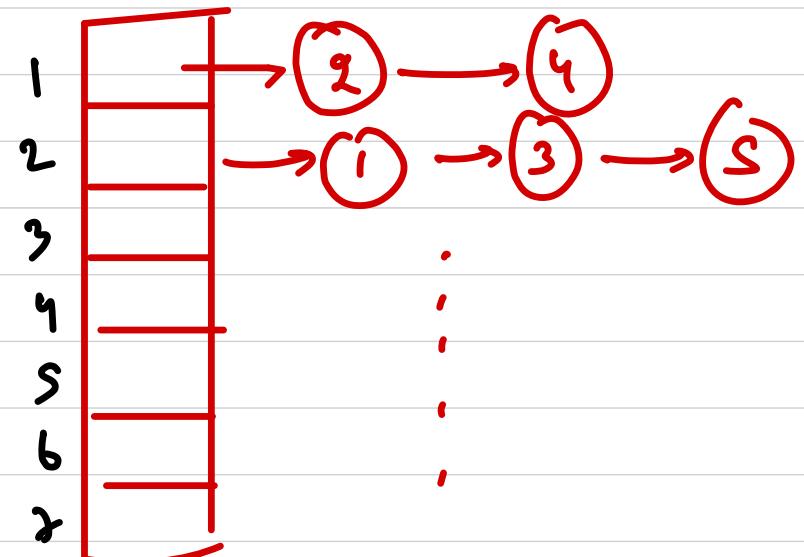
$\rightarrow O(V^2)$

Nahui

# adjacency list

$\{v, w\}$

Array of LL



Bucket array

$O(V+E)$  ~ space

# adjacency map / set

array of hashmp

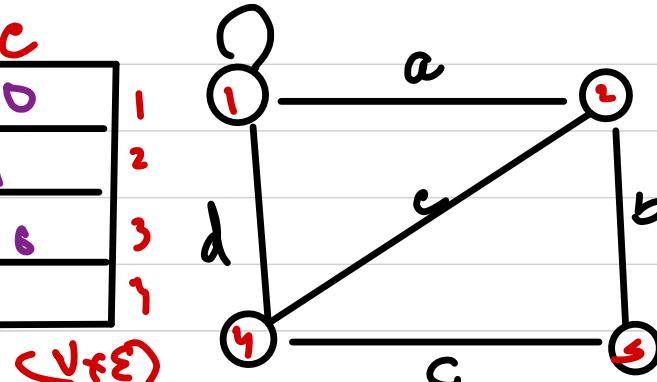
## Incidence

## Matrix

$S_{min}$

	a	b	c	d	e
1	1	0	0	1	0
2	1	1	0	0	1
3	0	1	1	0	0
4	0	0	1	1	1

$M =$



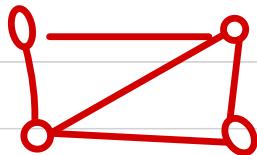
cols → edges  
rows → vertices

$M_{ij} = \begin{cases} 1 & \text{if the } i^{\text{th}} \text{ vertex belongs to the } \\ & j^{\text{th}} \text{ edge} \\ 0 & \text{else} \end{cases}$

columns → edges  
rows → vertices

Q What is a degree ??

Degree of a vertex in a graph  $G$ , is the total no. of edges incident / associated with it.

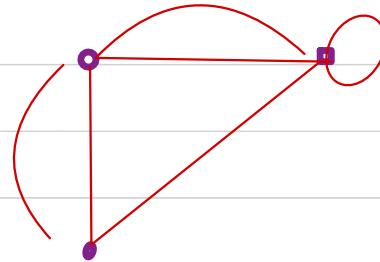


# directed  $\rightarrow$  indegree / outdegree  
graphs

outdegree  $\rightarrow$  no. of outgoing edges  
indegree  $\rightarrow$  no. of incoming edges

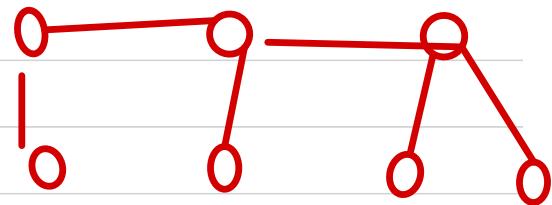
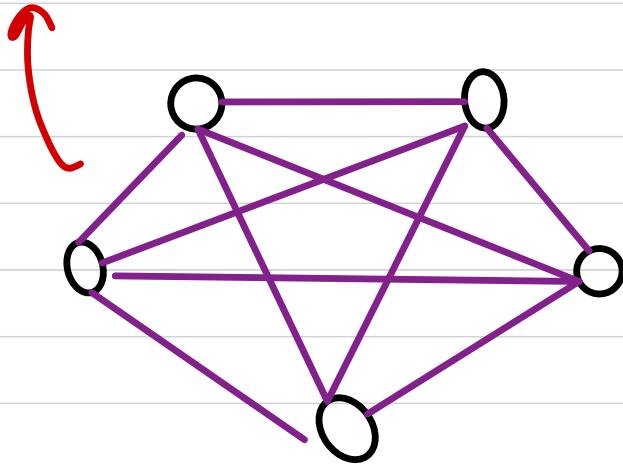
# Multi graph → an undirected graph with multiple edges btw vertices & <sup>e.g.</sup> loops allowed.

I multi graph



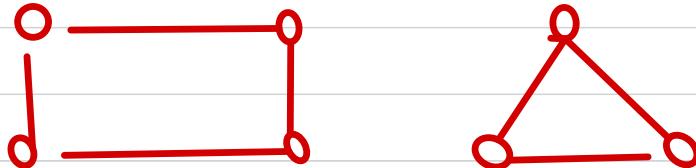
# Simple graph → An undirected graph in which both multiple edges & loops are not allowed.

# Complete graph → A graph in which every vertex is directly connected to every other vertex.



# Connected graph → A graph in which there is some path between 2 vertices but not necessarily direct

# Disconnected graph  $\rightarrow$  at least 2 vertices do not have  
a path to every other vertex:



# Component  $\rightarrow$  a subset of a disconnected/connected  
graph which is connected.

#path  $\Rightarrow$  A path  $p_n$  is a graph whose vertices can be arranged in some sequence such that



$$V = \{v_1, v_2, v_3, v_n\}$$

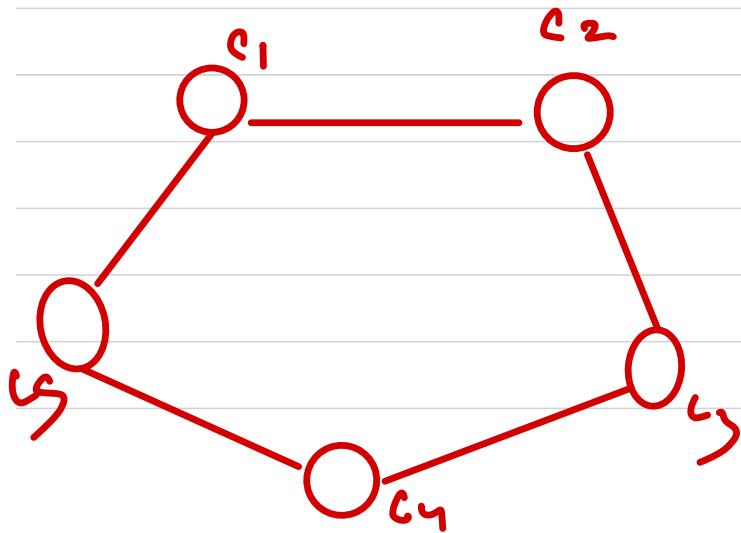
edge set of the graph is

$$E = \{v_i v_{i+1} \mid i \in [1, n-1]\}$$



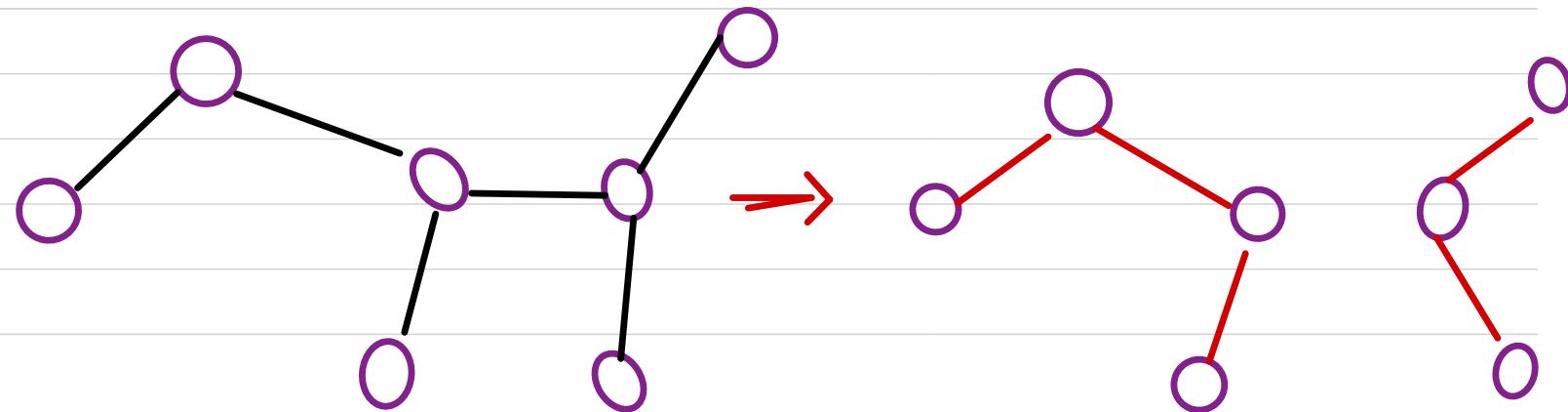
# Cycle  $\rightarrow$  A cycle  $C_n$  is a graph whose vertices can be arranged in a cyclic sequence. such that edge set is

$$E = \{v_i; v_{i+1} \neq i \in \{1, n-1\}\} \cup \{\underline{v_i, v_1}\}$$

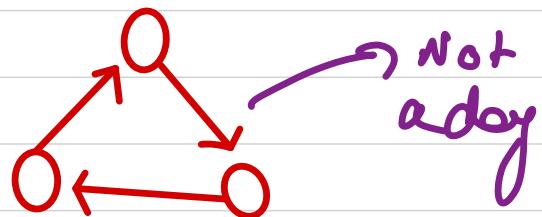


TREE → Tree is a connected graph with no cycles

forest → If we remove an edge from tree we get a forest which is collection of trees



DAG (directed acyclic graph)  $\rightarrow$



# How to read graphs

As graphs are non-linear, we need some mechanism to read graphs.

## Graph traversal

→ Depth first

→ Breadth first



## Depth first search

TC  
?

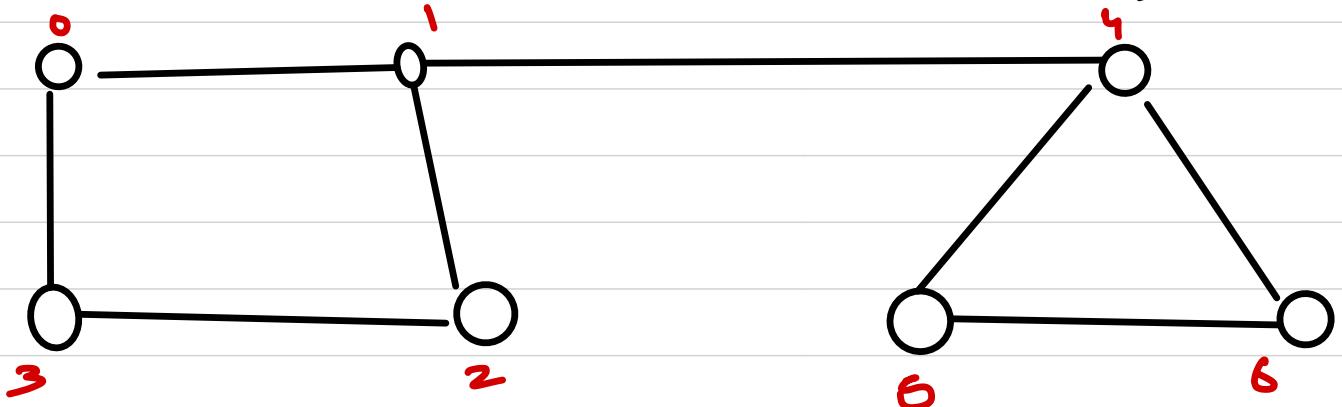
$O(V+E)$

\* Motivation problem:

~~Q.~~ Given a graph calc all paths between two vertices

OR

~~Q.~~ Given a graph check whether there is a path between 2 vertices.



is there a path from 0 - 5

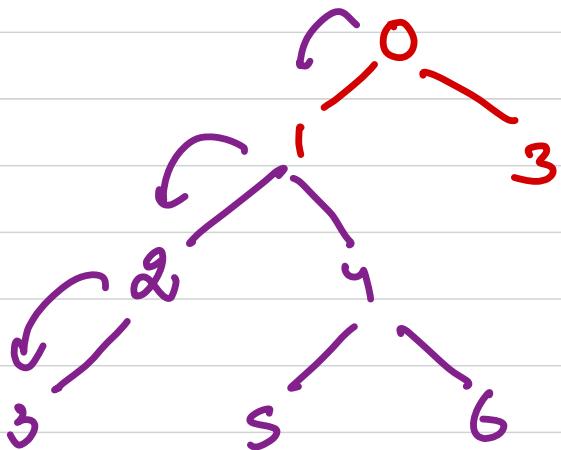
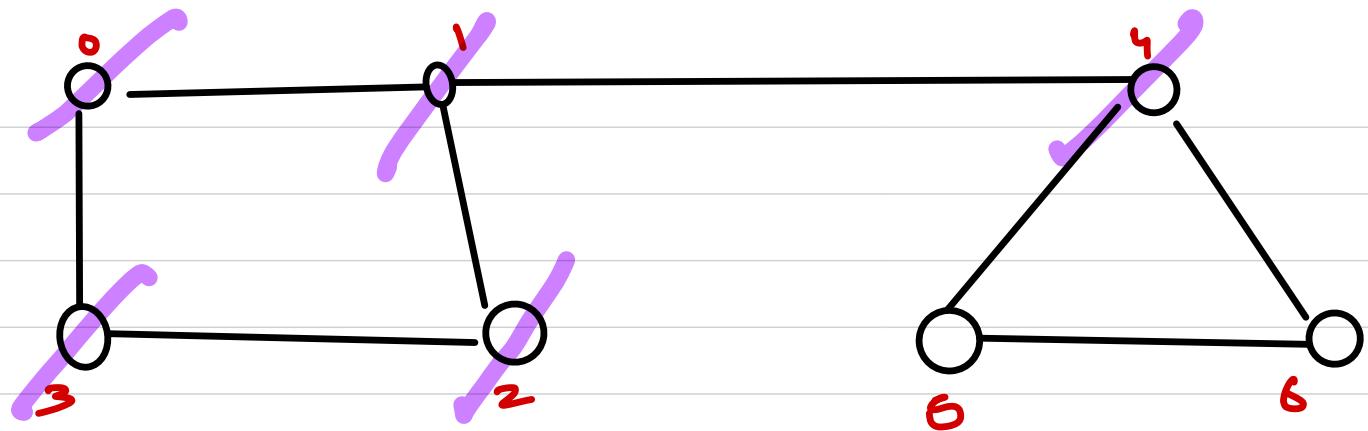
What is the most easiest query for this? ?

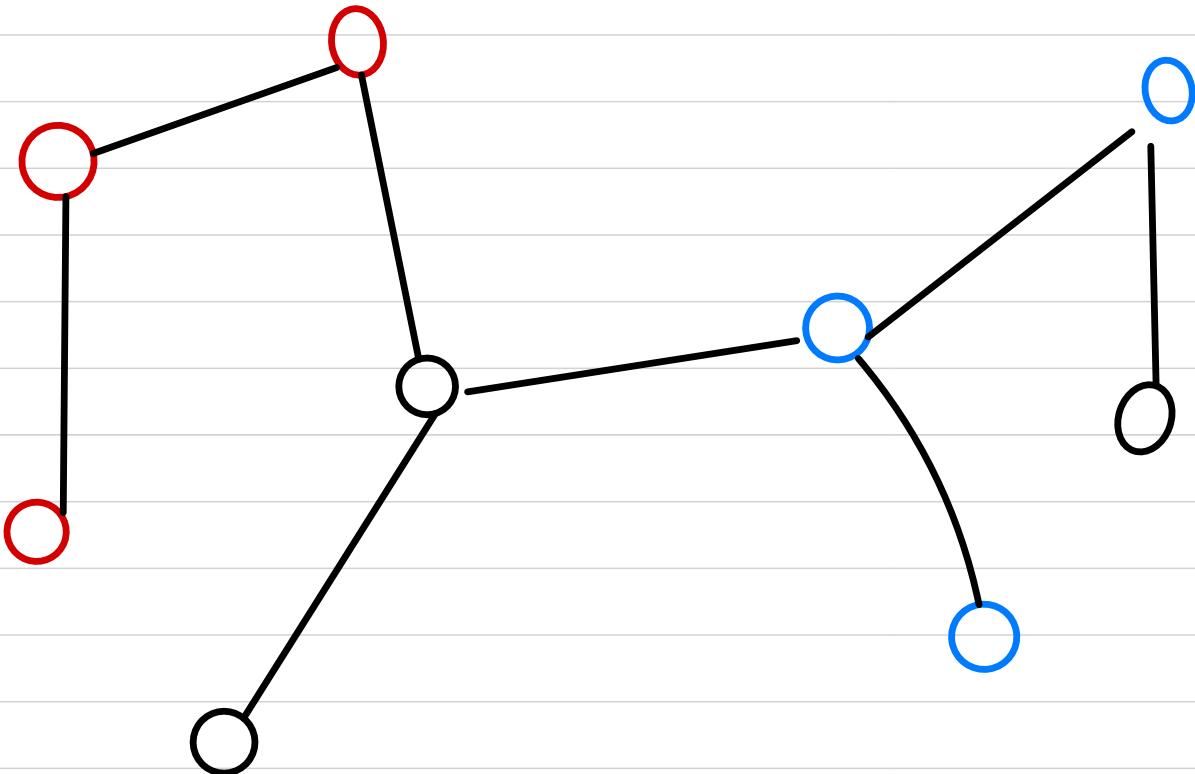
→ neighbours

$$f(u, v) = \begin{cases} f(n_1, v) \\ \text{or} \\ f(n_2, v) \\ \text{or} \\ f(n_3, v) \\ \vdots \end{cases}$$

whether there is  
a path from  
u to v.

where  $(n_1, n_2, n_3, \dots) \in$  neighbors of u.





# Proof that a tree with  $n$  nodes has  $n-1$

~~edges.~~

P MI  $\rightarrow f(n)$   
no. of edges for  
 $n$  nodes

$$f(1) \rightarrow 0$$

$$f(2) \rightarrow 1$$

To prove  $\rightarrow f(n+1)$

We assume function works for  $f(1) \rightarrow n-1$

Total no. of edges  $\rightarrow$  no. of edges required for  $(n+1)^{th}$  node  
+  $(n-1)$  edges

Every node that will be added to a tree requires

only 1 node.

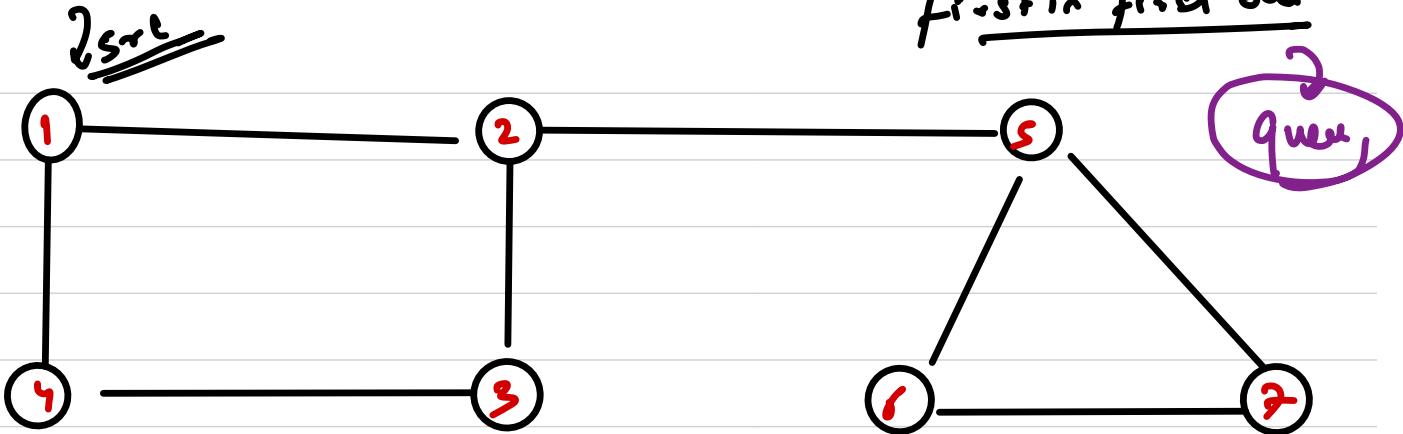
$$\boxed{f(n+1) = (n-1) + 1 = n}$$

equivalent  
to level order  
traversed

## Breadth First Search



Say we have an arbitrary src node then all the nodes  
shortest (by edges)  
with same distance from src node are said to be  
on the same level.



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

Breadth first  
traversal

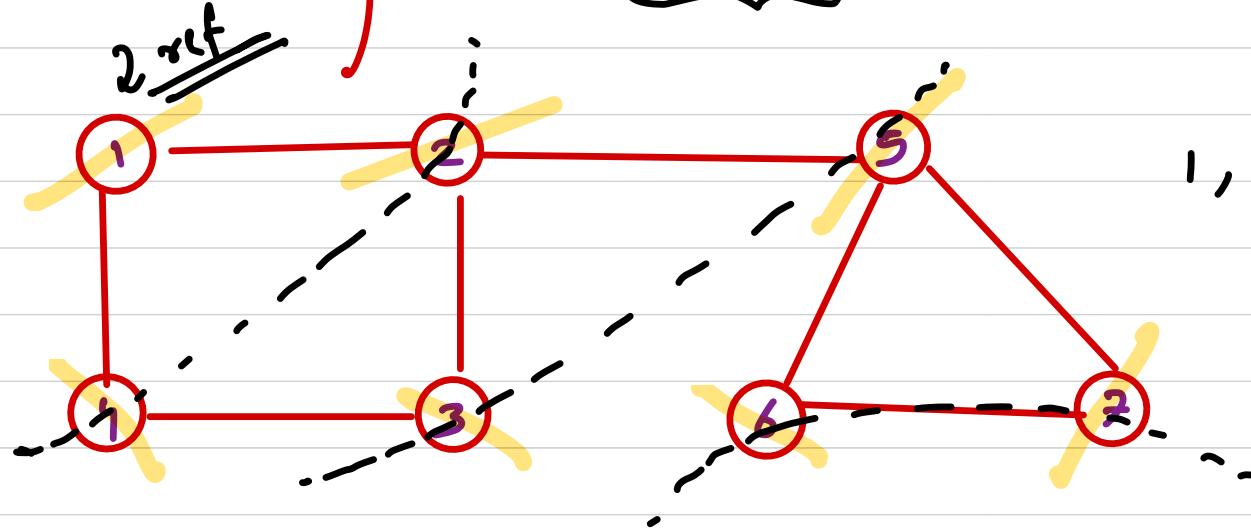


→ queue

unreliable

## Breadth First Search

$f(f)$



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



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

X X X X X X X X X X

5 queen

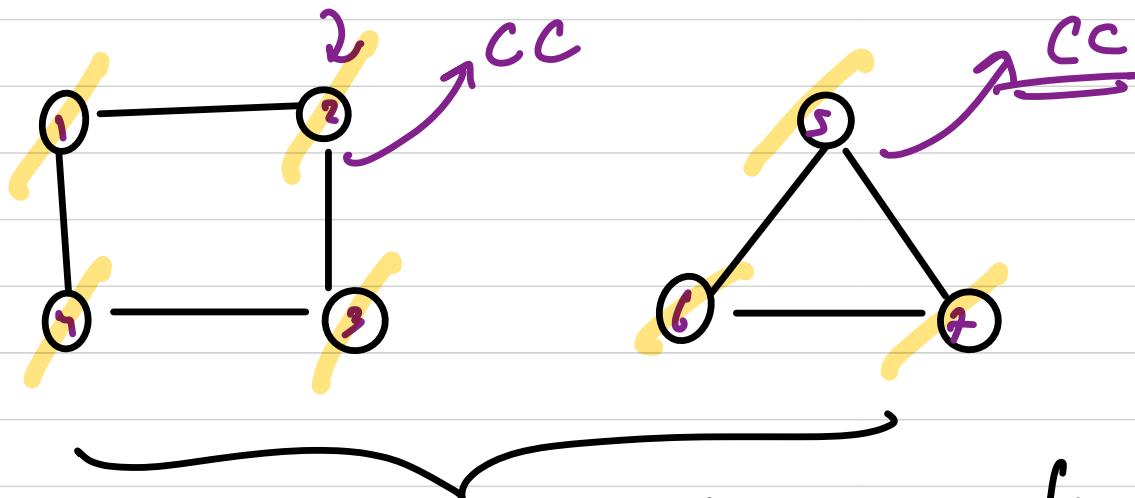
Time  $\rightarrow O(V+E)$   
Space  $\rightarrow O(V)$

So in breadth first search, when we travel from  $\gamma$  of node  $\rightarrow$  any node then it always consider shortest path.

Connected

Components

Dfs / Bfs



Disconnected  
graph

vis = {}

iterate on all  
possibly vertices

```
for (i = 1; i <= n; i++)  
if (!vis[i])  
dfs(i)
```

So the no. of times we will call dfs will be  
the count of CC.

# Journey to the moon

pair of astronaut  $\alpha_1, \alpha_2 \rightarrow$  Then  $\alpha_1, \alpha_2$  are  
from same Country

Choose  $\rightarrow 2$

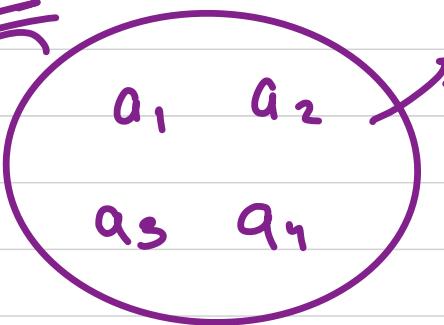
Say, total no. of ast =  $n$

$${}^n C_2 \rightarrow \frac{n!}{2!(n-2)!} \Rightarrow \frac{n(n-1)}{2}$$

? 2

$\langle a_1, a_2 \rangle$

~~group~~

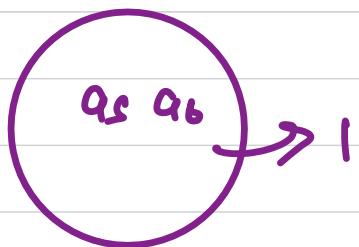


$\langle a_2, a_3 \rangle$

$\langle a_3, a_1 \rangle$

$\langle a_5, a_6 \rangle$

${}^4 C_2 \rightarrow \frac{4 \times 3}{2} = 6$



$${}^6 C_2 \rightarrow \frac{6 \times 5}{2} = 15$$

$$15 - 6 - 1 \\ \Rightarrow 8$$

# Permutation Swaps

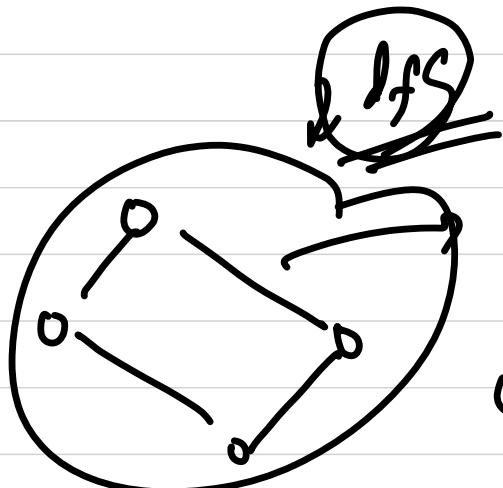
[ 11, 21, 31 ]  
  \underbrace{  \underbrace{  }

: d 4   1   2   ,   7

A > 1   3   2   4

B > 1   4   2   3

1, 2  
2, 3



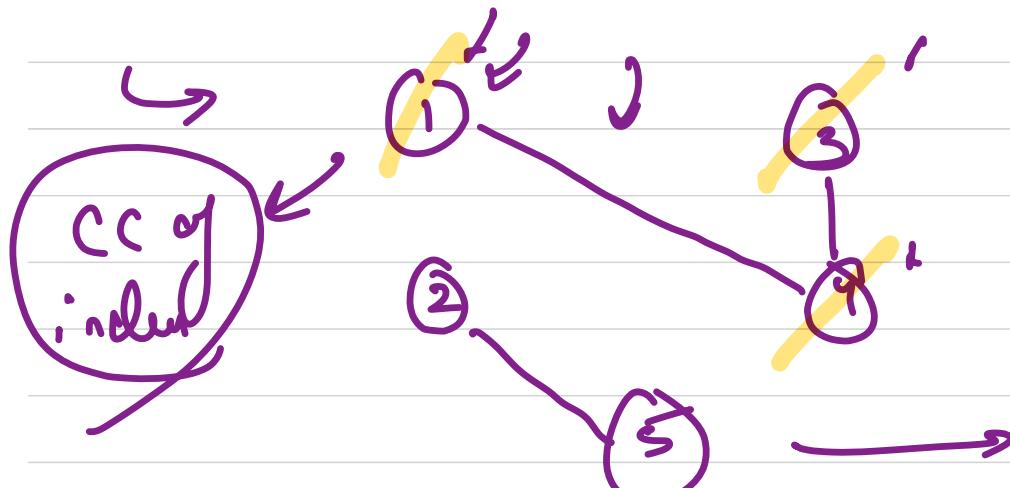
(3, 4) ← index

those indices  
are in the same  
connected  
component

idn 1 2 , 1 5



B  $\rightarrow$  1 5 2 3 4



relativ  
 $P \rightarrow [1, 2, 3]$

$q \rightarrow [1, 3, 2]$

$P \rightarrow [q, s]$   
 $q \rightarrow [s, r]$

$i d_1$  1 2 3 4 5

$(2, 5)$

4 1 3 4 3 2

$(3, 4)$

5 1 4 2 3 5

$(1, 4)$

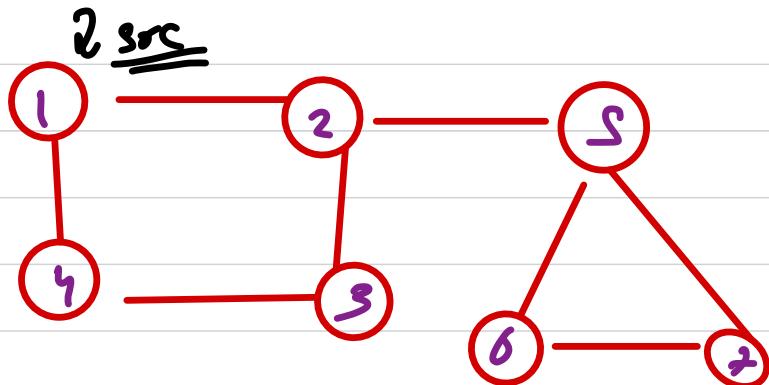


$\rho \rightarrow [1, 3, 4]$

$q \rightarrow [1, \cancel{2}, \cancel{2}, 3]$



~~Q~~ Given an <sup>unweighted undirected</sup> graph, and a source node calc the length of the shortest path to all the other nodes from the source node.



→

0	1	2	1	2	3	3
1	2	3	4	5	6	7

↑ output

BFS

BFS

Bfs traverse the nodes based on shortest

path

Q Given a graph, a src and a destination,  
find all the nodes which are a part of  
at least 1 shortest path.

O(v + e)



$$\text{dist} \rightarrow (s, d) \rightarrow x$$

$$s[i] + d[i] = x$$

$$a + b = x$$

$$b = x - a$$

Rotten

Oranges

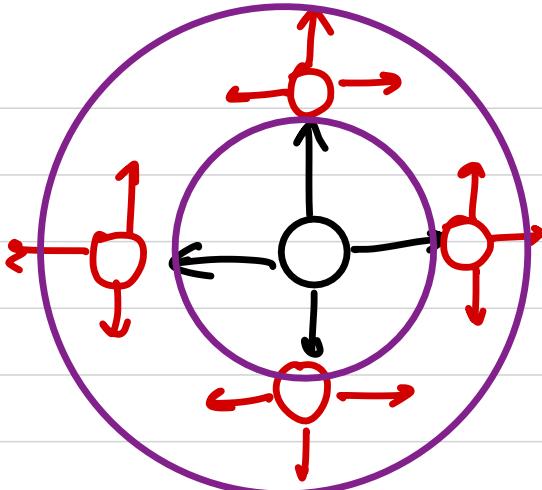
6	2	X2	Y2	2
2	Y2	X2	0	0
0	21	0	X2	2
0	0	2	2	2
0	0	0	0	0

(2D array)

m x n

-1 → when no  
other oranges  
can rot a  
orange

$t = t'$  → all oranges which  
are rotten will offset their  
neighbours at same time



BFS

queue

vis

$(i, j)$

Multi Source BFS

$O(nm)$

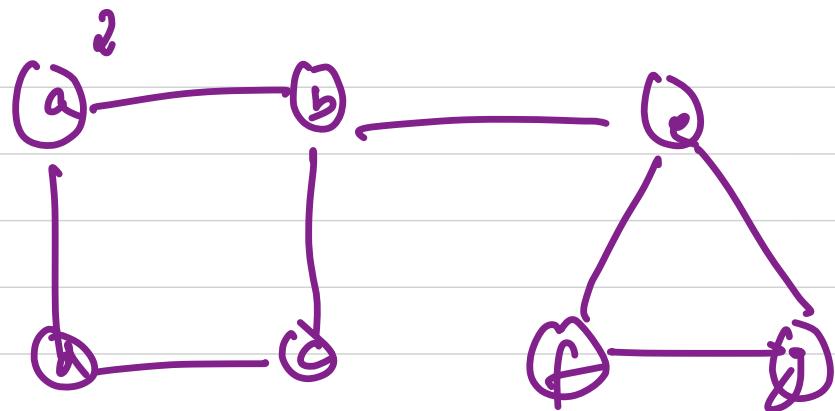
$(i+1, j) \quad (i-1, j) \quad (i, j-1) \quad (i, j+1)$

all this will be marked  
if there is a fresh orange

$\begin{array}{c} \uparrow \\ \leftarrow 2 \rightarrow \\ \downarrow \end{array}$

$\begin{array}{c} 1 \\ \leftarrow 2 \rightarrow \\ \downarrow \end{array}$

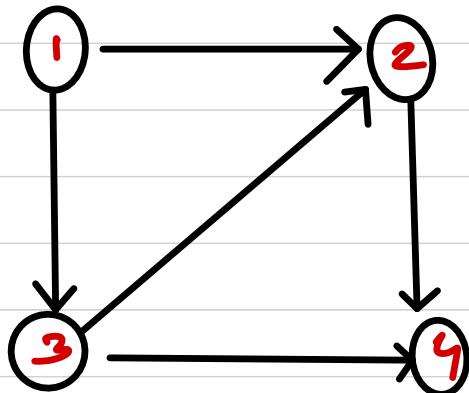
$\begin{array}{c} 9 \\ \leftarrow 2 \rightarrow \end{array}$



~~Algorithm fig~~

Longest Path

Dynamic Prog.



→ The longest directed path can start from any node.

$f(v)$

1D DP  
=

returns the length

of the longest path

that starts with v.

when  $n_1, n_2, n_3, \dots, n_m$  are  
the neighbours of v

ans  $\max(f(1), f(2), f(3), \dots, f(n))$

$1 + \max$

$f(n_1)$

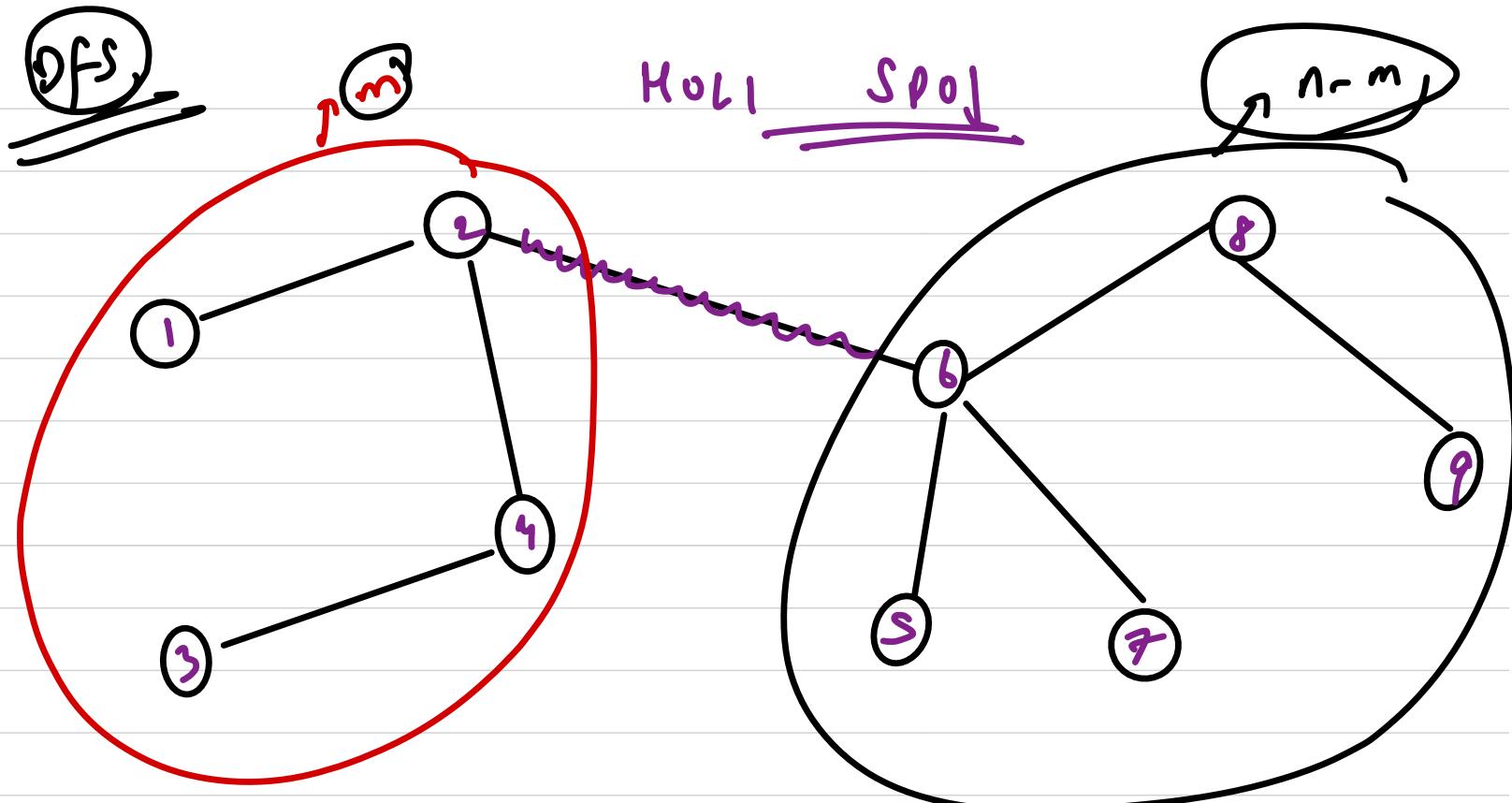
$f(n_2)$

$f(n_3)$

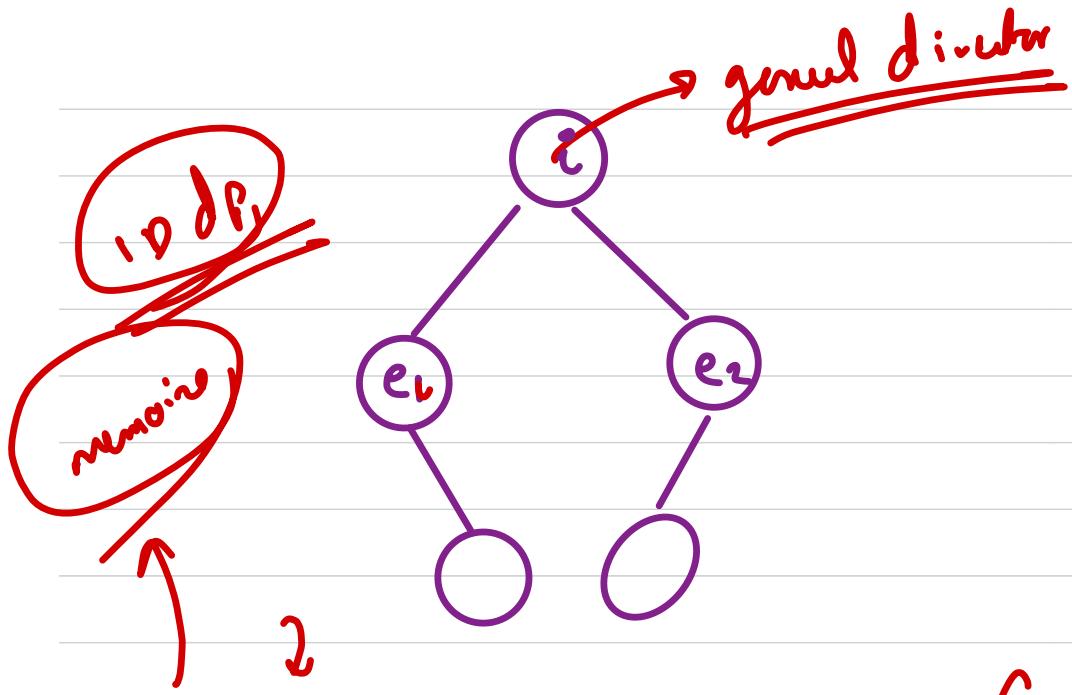
⋮

$f(n_m)$

Base  
case  
length  
0



$$\underbrace{g(n-m, m)}_{\text{ }} \times \overrightarrow{\text{wt}}$$

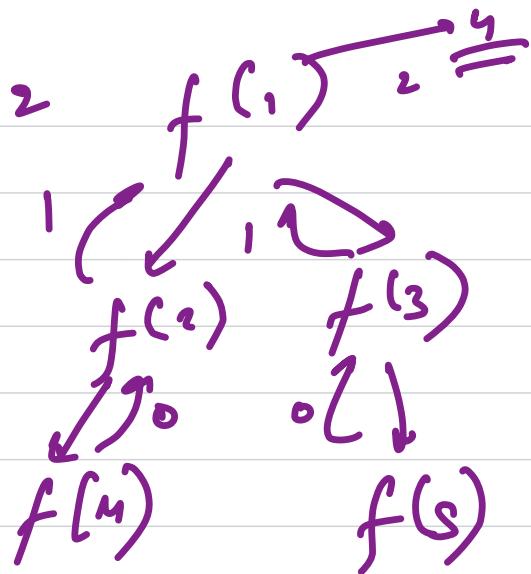
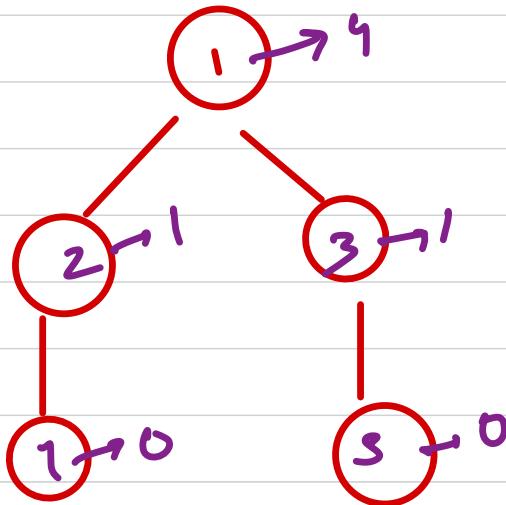


$\downarrow$

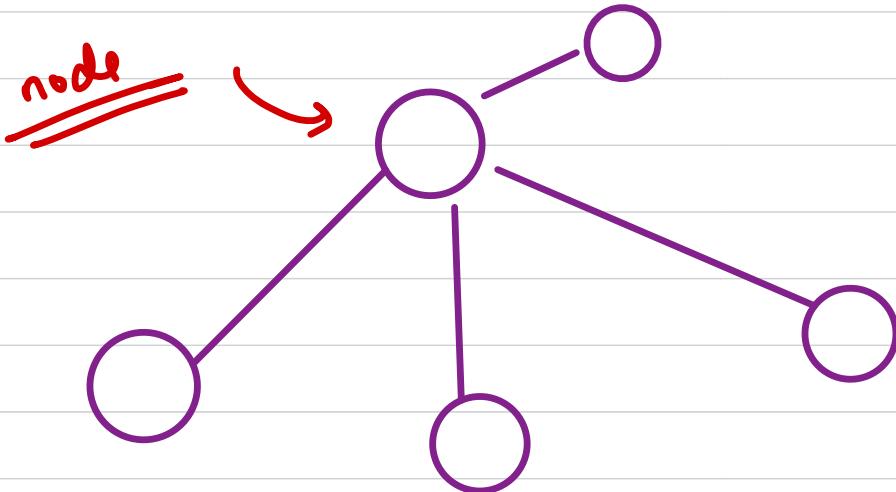
$f(i)$

no. of subordinates  
under employee  $i$

$$= \sum (1 + f(e_i))$$



Independent Set → Attoder



color  $\begin{matrix} \text{red} \\ \text{white} \end{matrix}$   $\begin{matrix} \text{B} \\ \text{W} \end{matrix}$

Black → cast  
White → no cast

No, two adjacent vertices should be black

$f(\text{node}, c) \rightarrow$  these two parameters can  
uniquely identify our state

$c \rightarrow \text{constraint}$

$f(\text{node}, 0) \rightarrow$  # of ways to color subtree  
rooted at node

as  $C \rightarrow 0$  then node can be w or b

$f(\text{node}, 1) \rightarrow$  # of ways to color subtree rooted  
at node and node is under constraint

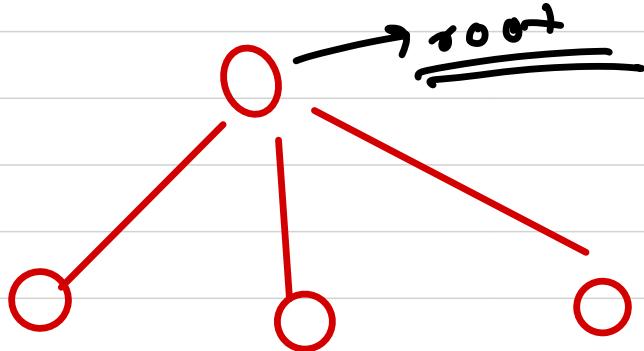
$$\begin{cases} f(\text{node}, 0) = \pi f(\text{child}, 0) + \pi f(\text{child}, 1) \\ f(\text{node}, 1) = \pi f(\text{child}, 0) \end{cases}$$

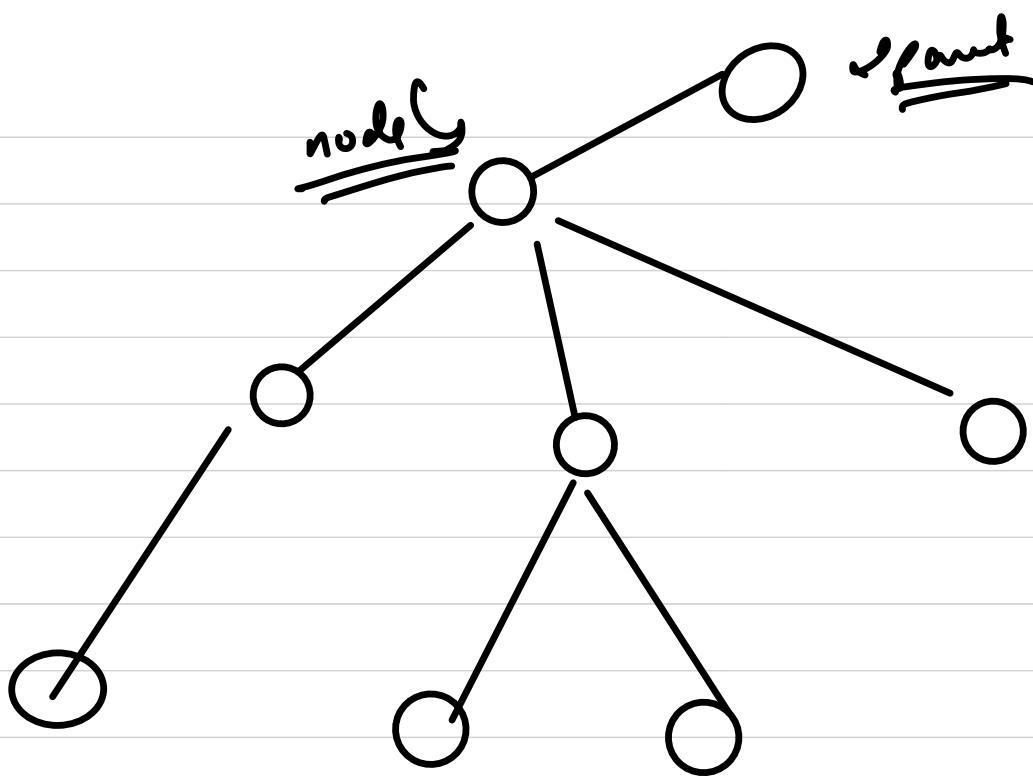
final ans  $\rightarrow f(\text{root}, 0)$

LeetCode

# Binary Tree Cameras

Brute force





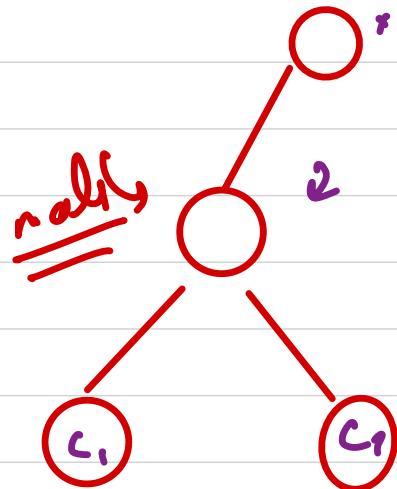
$f(\text{node}, \text{cam}, \text{par cam})$

↓

return the min number of cans

to cover the subtree rooted at node.

Final ans  $\rightarrow \min(f(\text{root}, 1, 0), f(\text{root}, 0, 0))$



for current node , we decided to have  
or can the parent is not have a car

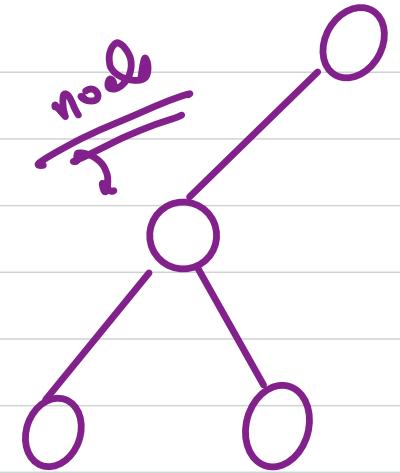
$$f(\text{node}, 1, 0)$$

for your cars

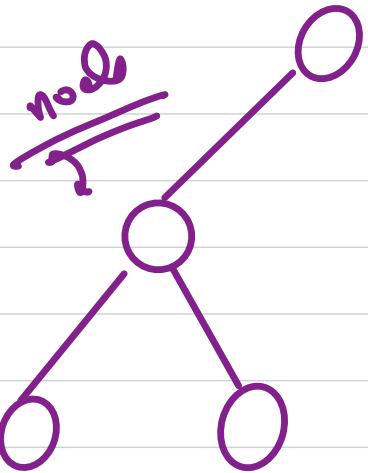
$$f(\text{node}, 1, 0) = 1 + \min(f(c_1, 1, 1), f(c_1, 0, 1)) + \\ \min(f(c_2, 1, 1), f(c_2, 0, 1))$$

when node isn't key can but parent  
has it

$$f(\text{node}, 0, 1)$$



$$\begin{aligned} f(\text{node}, 0, 1) = \min & (f(c_1, 0, 0), f(c_1, 1, 0)) + \\ & \min (f(c_2, 0, 0), f(c_2, 1, 0)) \end{aligned}$$



$$f(\text{node}, 0, 0) =$$

$$f(\text{node}, 0, 0)$$

so dp

one of the children are bounded to have a can.

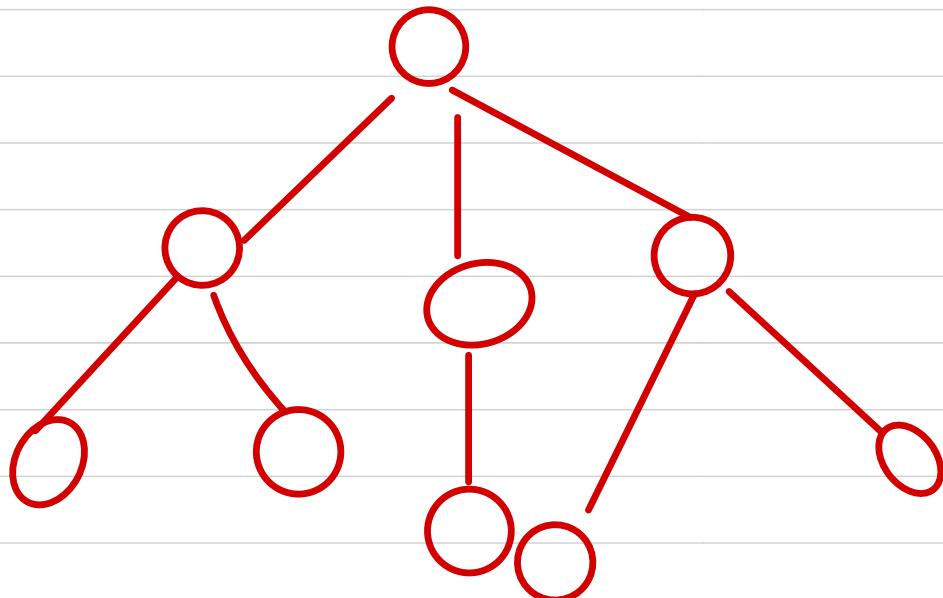
$$\begin{cases} 
 f(c_1, 1, 0) + \min(f(c_2, 1, 0), \\ f(c_2, 0, 0)) \\ 
 f(c_2, 1, 0) + \min(f(c_1, 1, 0), \\ f(c_1, 0, 0))
 \end{cases}$$

ans =  $\infty$

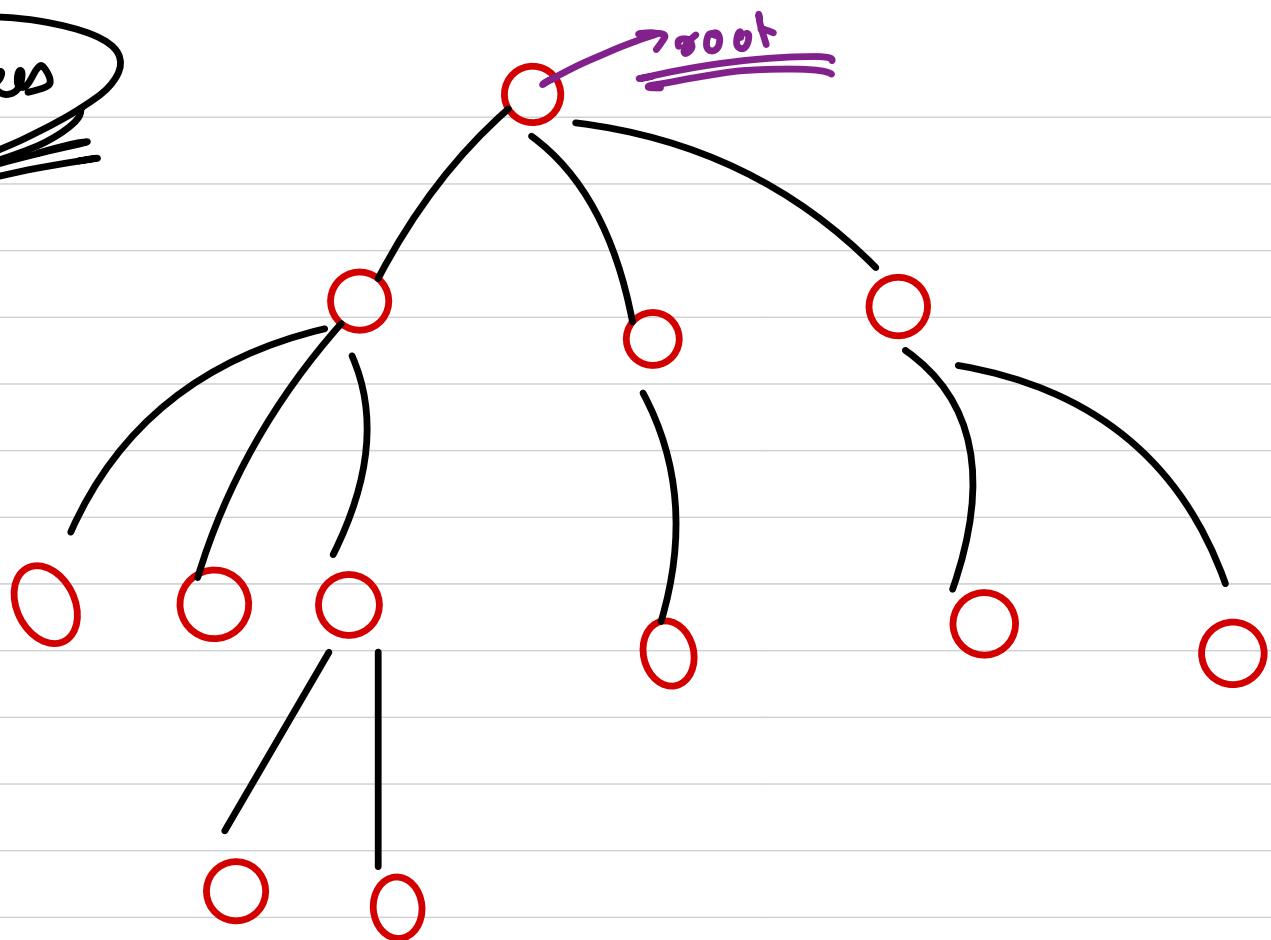
fam tree

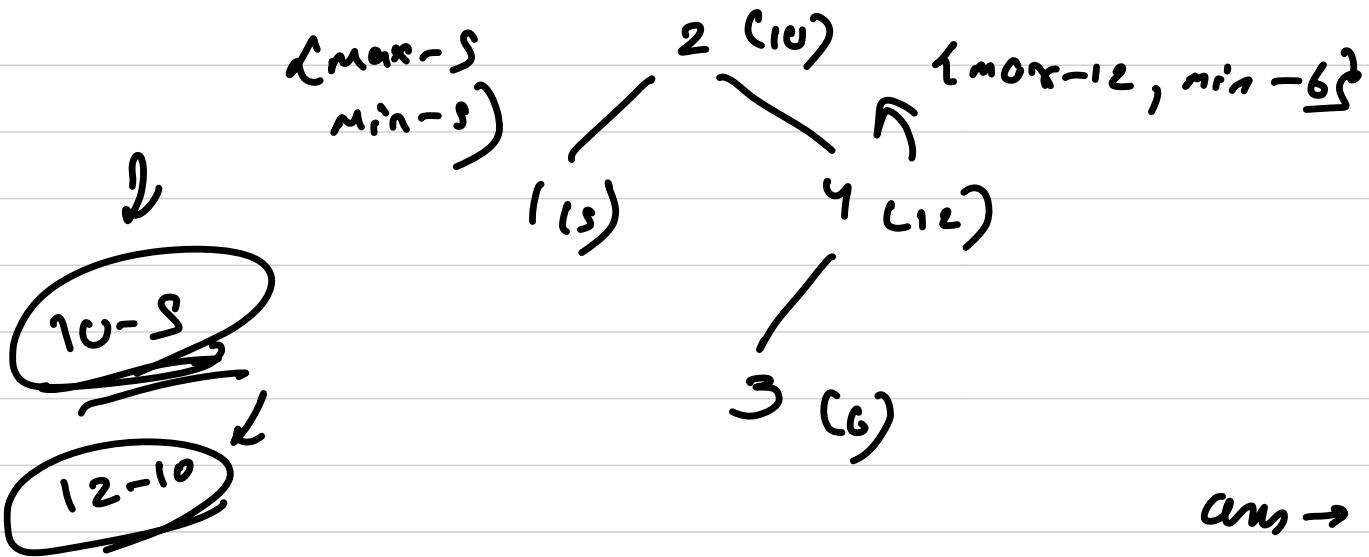
(Codichef)

max  
min



Dynam trees





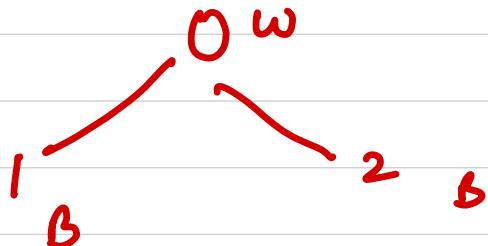
ans → ~~6~~  
 6  
 ===

#applemans and tree → codforces

Divide this tree into arbitrary  $k$  components

such that each component has exactly one black

node-



$$0 \leq k \leq n$$

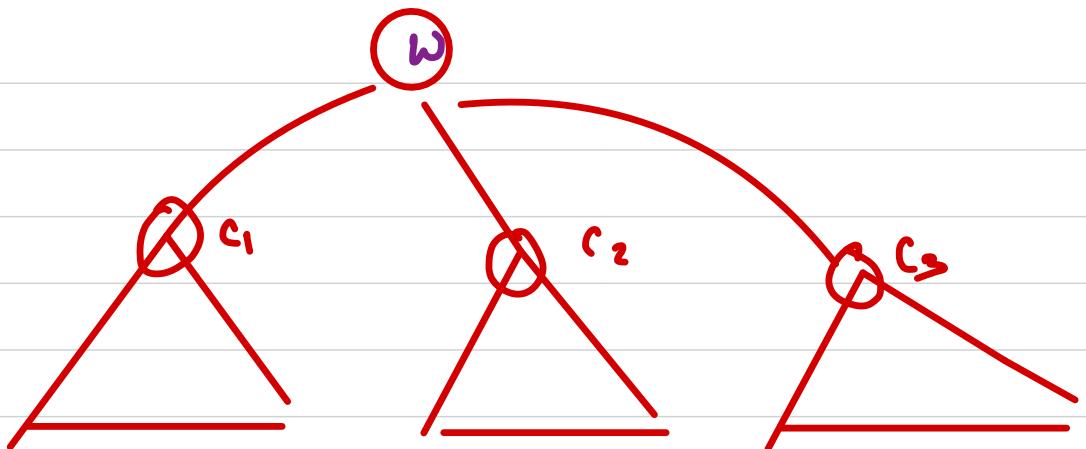
tree →  
 $(n^k)$

every node will have different choices  
based on colors of node      inSubtree

$f(\text{node}, c)$        $\downarrow$   
count  
of black  
n-nodes

$f(\text{node}, 0) \rightarrow$  # of ways subtree rooted at node  
has no black node vertex

$f(\text{node}, 1) \rightarrow$  # of ways for subtree rooted at node  
& has 1 black vertex  $\equiv$



$$f(\text{node}, i) = \pi(f(c_1, 0) + f(c_1, 1))$$

$$f(\sim \text{node}, 0) = 0$$

$$f(\text{node}, 0) = \pi(f(c_1, 0) + f(c_1, 1))$$

$$f(\text{node}, 1) = \sum f(c_{1,1}) \times \frac{f(\text{node}, 0)}{f(c_{1,0}) + f(c_{1,1})}$$

~~Qn~~ You're given a set of inequalities .  
~~Eg~~  $(a < b \quad c > d \quad b < d \dots)$

You've to find that whether for any set of int values for the variables, can be solve the inequality .

## Dependency      Resolution

- ↳  $a < b$ ,  $c > d$ ,  $b < d$
- ↳ package of a software
- ↳ course structure

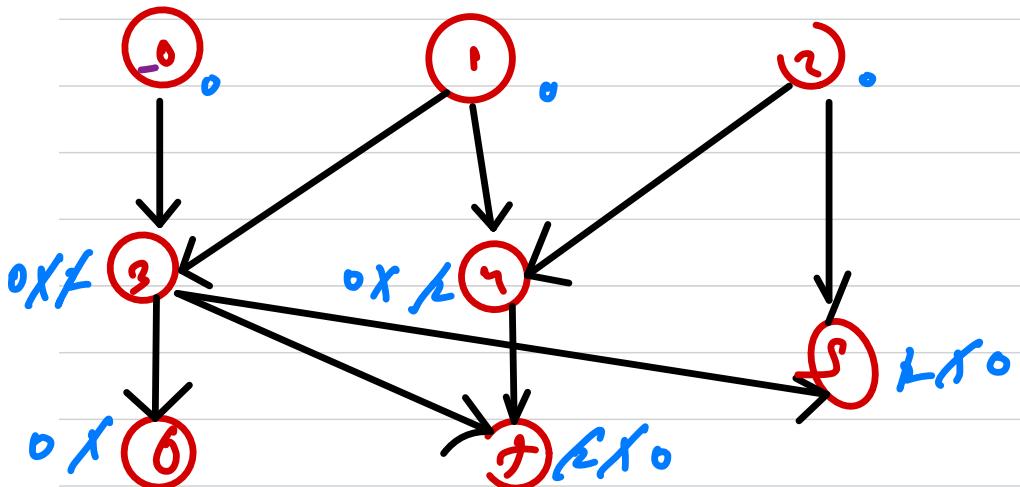
→ directed graph

Can be visualized  
in the form of  
a graph.

In order to resolve the dependency graph we can  
use an algorithm called as topological sorting

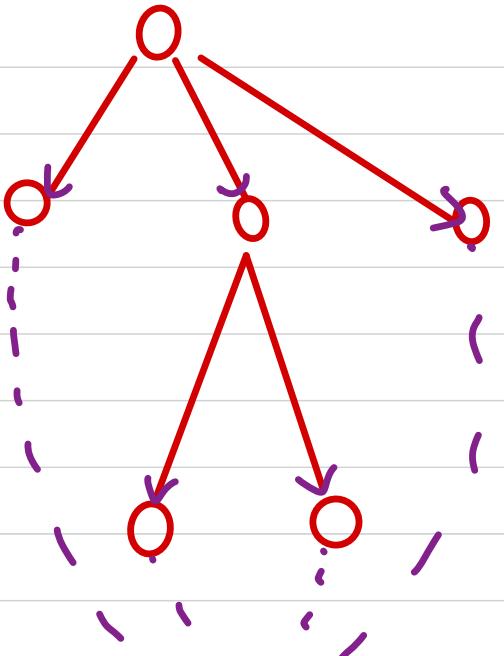
Kahn's algorithm  $\rightarrow$  .BFS based topological sort

queue

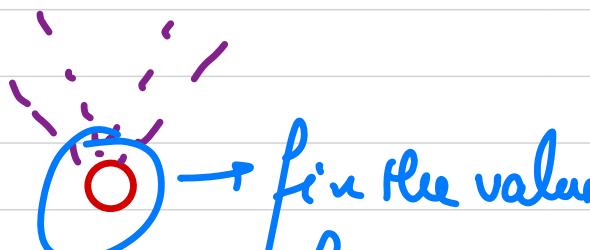


5, 2, 3, 1, 8, 2, 4, 6

0, 1, 2, 3, 4, 5, 6, 7



0,1,2,3



fix the value  
here i.e Node N.

0,1,2,3

$f(\text{node}, \text{xor})$

~~Ques~~ Given a grid with char [L, R, U, D]

on each cell of grid. You're standing at top left of the grid. From each cell you can go in the direction written on the cell for ex L → Left.

Decide if it is possible to reach the bottom right corner or not. Dimension →  $M \times N$

Explain Space  $\Rightarrow$  O(1)

$m, n \leq 10^3$

↳ There is only one case for not ready the bottom right.

↳ if there exist a cycle



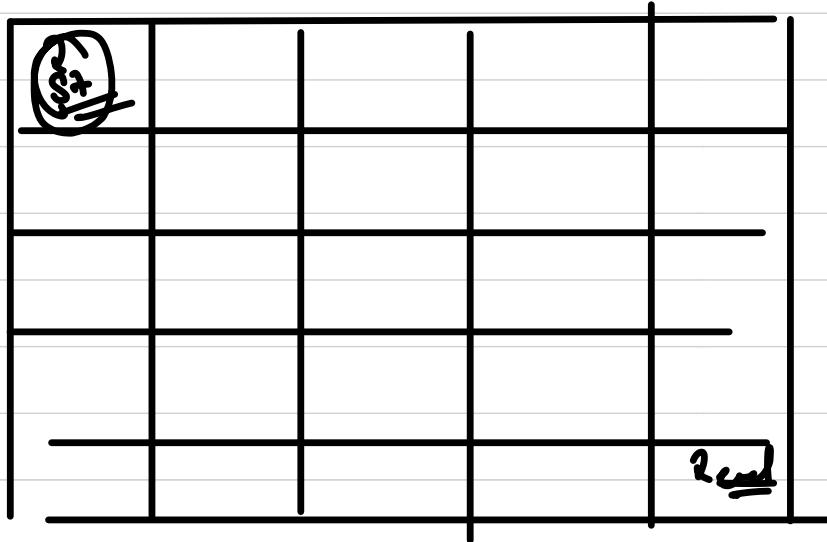
$1 \rightarrow 1x$

$2 \rightarrow 2x$

# Say we can swap the characters of the cell with any other then we want. Cost of changing char is 1 unit.

Calc the min cost to reach bottom right.

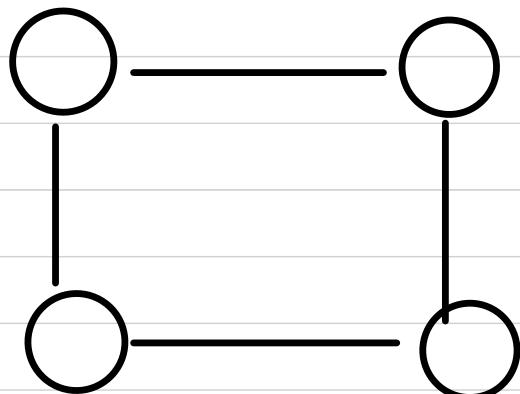
I cant



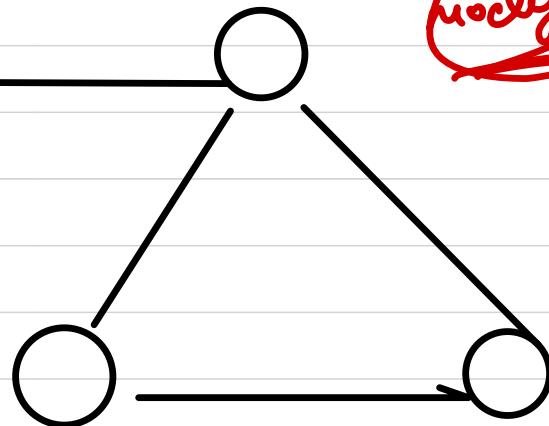
on this weighted graph calc the shortest path

Given a graph & a src and a dest node  
Find the shortest path from src → dest

? src

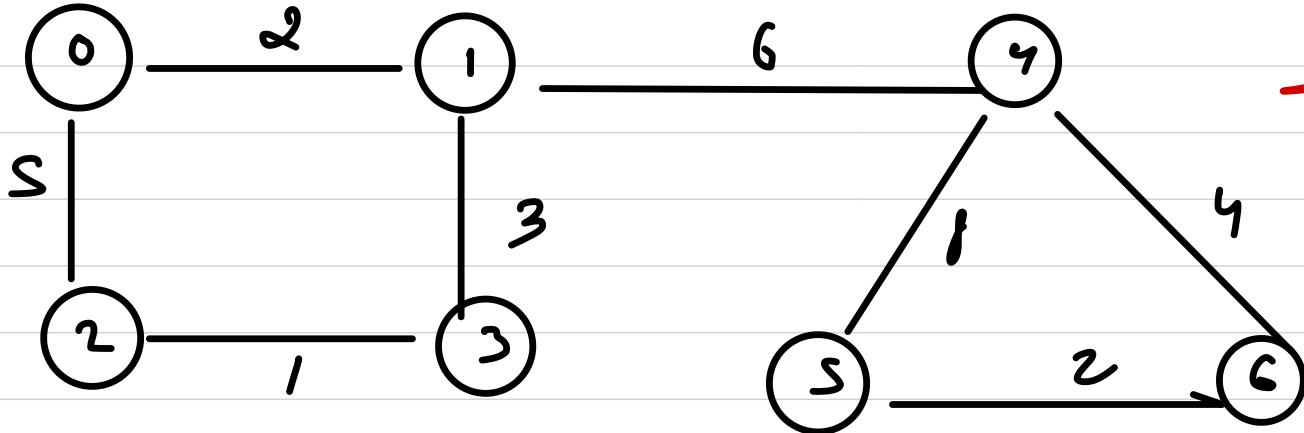


BFS  
→ modify



short

28-0



Dijkstra's

Priority Queue



node	parent	wt
0	-	0
1	0	2
2	0	5
3	0	6
4	0	8
5	0	9
6	0	10

Joint

Topf

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

Strongly Connected Components

directed graph



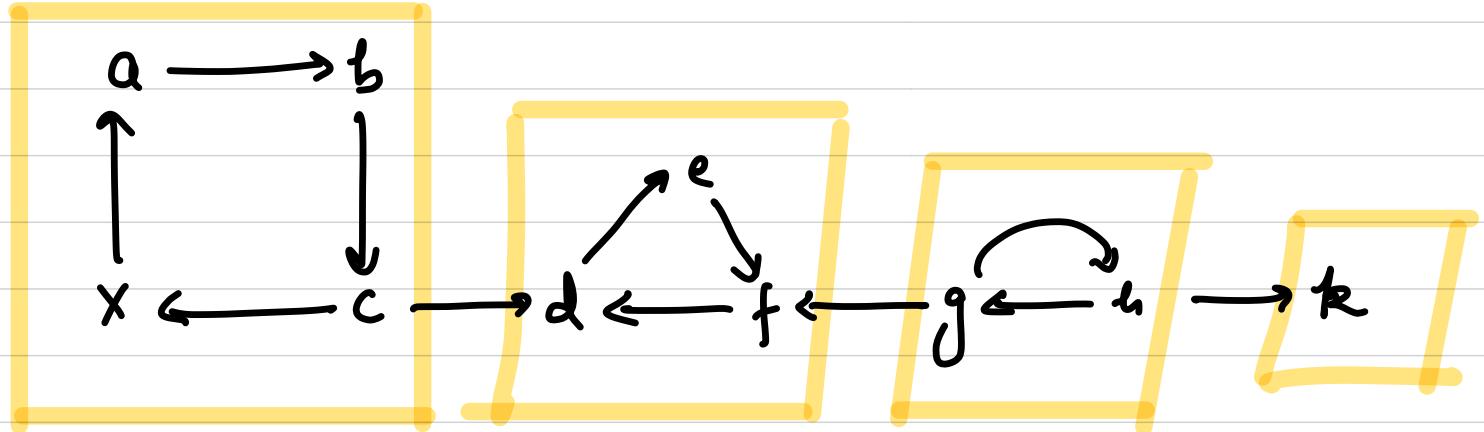
We have a path

from a to b

but not vice-versa

Within a component in a SCC

every vertex should have a path  
to other vertex in a directed graph



How can we find SCC ??



for finding normal cc in an undirected graph,  
we used to take help of dfs / bfs

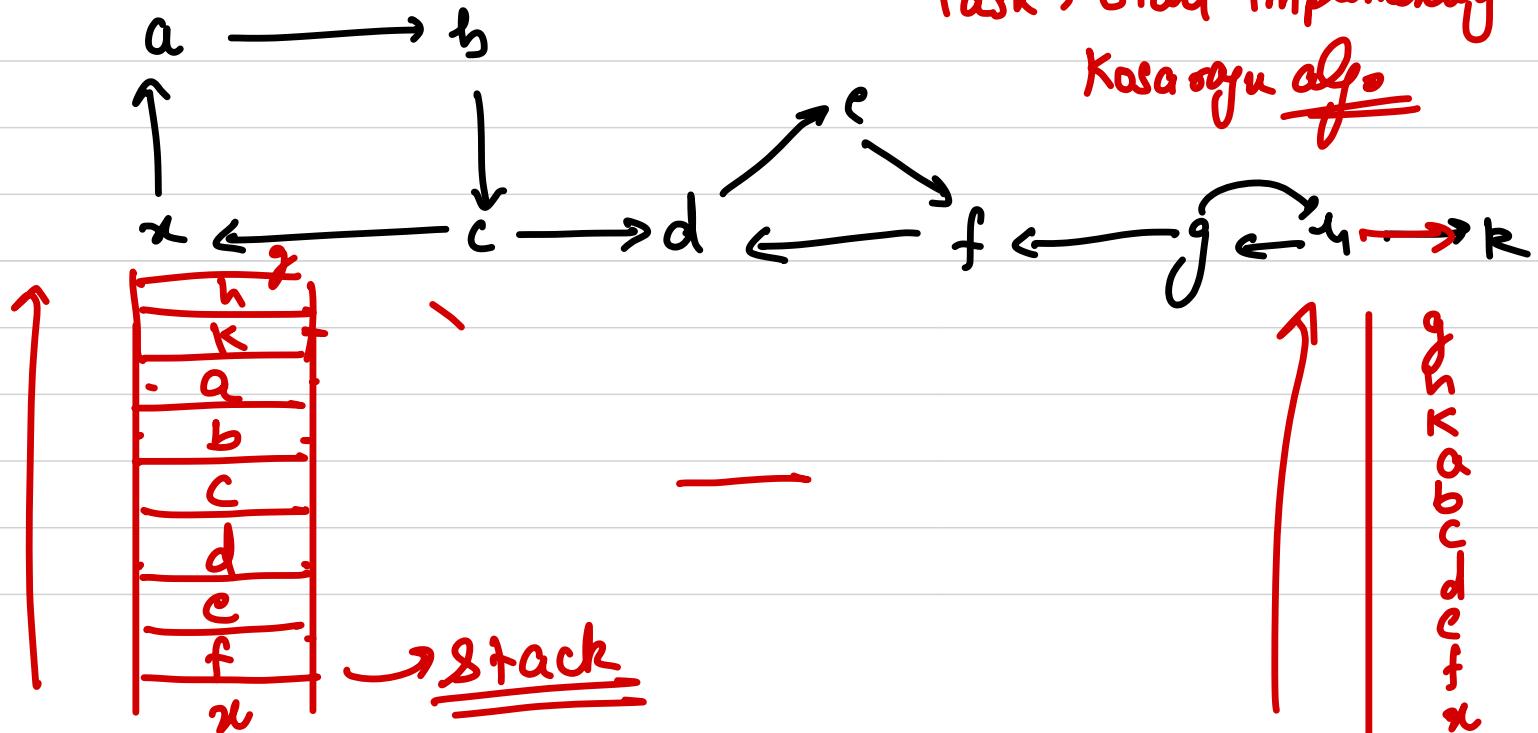
# Kosaraju's

# Algorithm

visited

a, b, c, x, d, e, f  
g, h, v

Task → Start Implementing  
Kosaraju algo



We will start bfs/dfs from an unvisited node.

If we have touched all possible nodes in a go

then we restart with any other unvisited if

it is left.

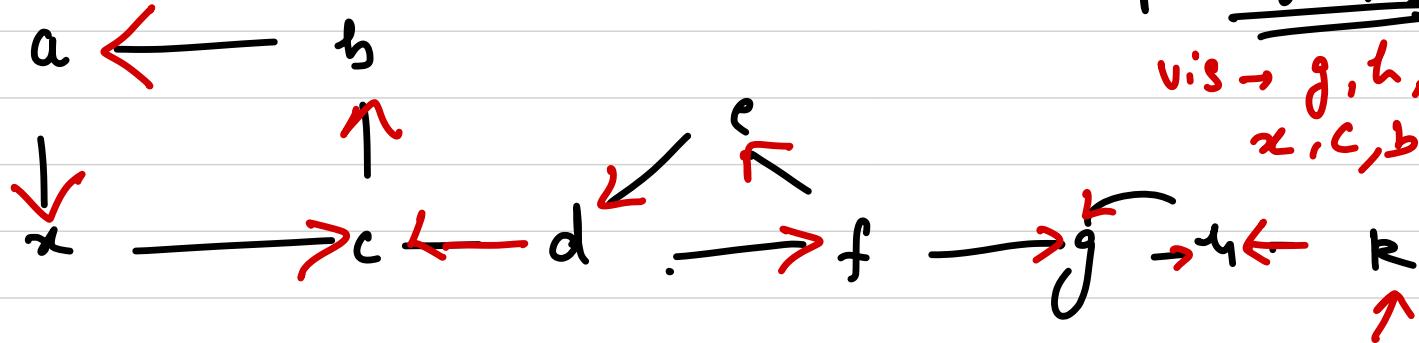
Do dfs & store vertices in the stack as per their finish order.

In the phase 2, take transpose of the graph

Start dfs on the transposed graph from the node at the top of stack:

2 transposed graphs



v.i.s → g, h, K, a,  
z, c, b, d, f, i

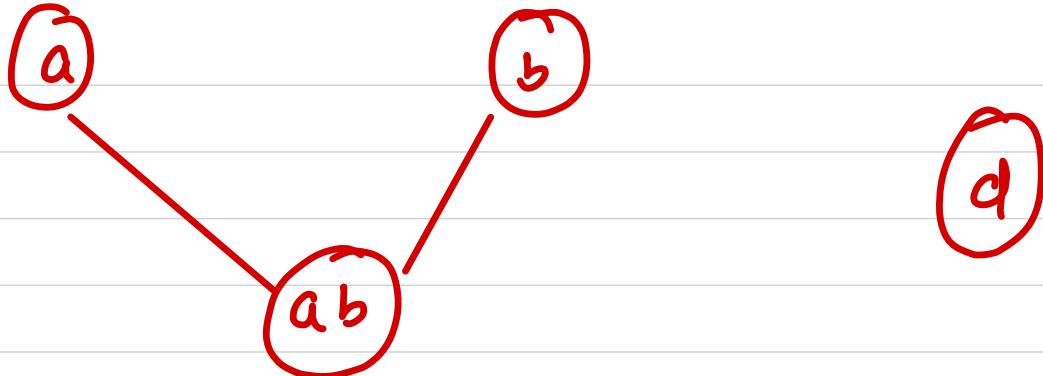
gh

K

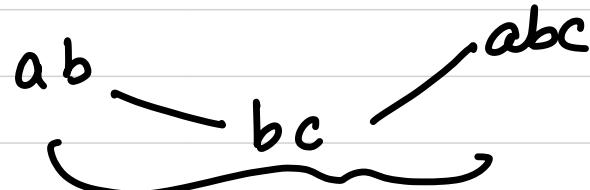
a z c b

d f c

=>



2<sub>1</sub>      ofs / Bfs / Dsu



Same component

one component

$10^5$

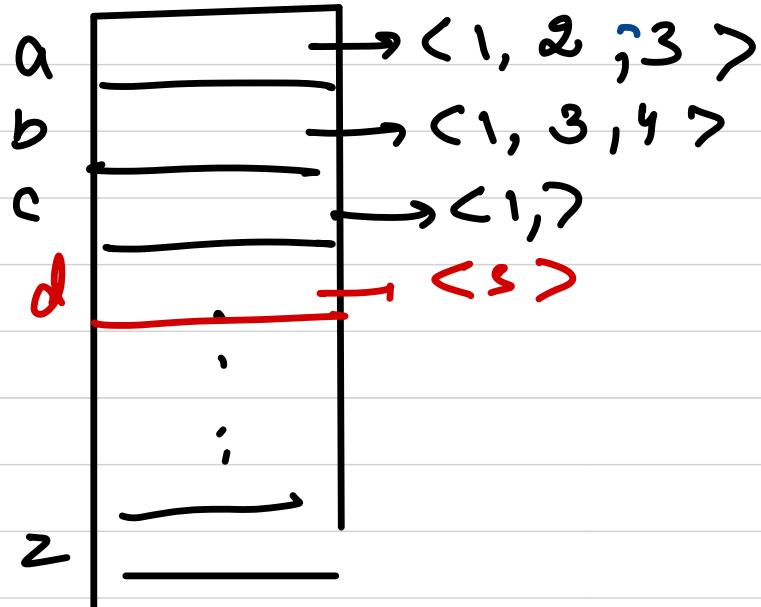
$a b c$

$a b$

$b$

$a b$

$a d$



$4 \swarrow 3 \nearrow 2$   
 $4 \swarrow 1$

$s$

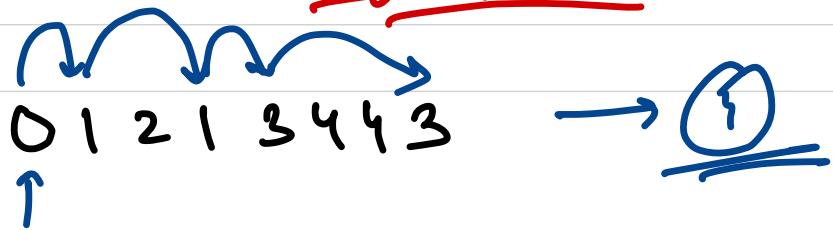
Q7 Given a string of digits of length N. ( $NS^{10^5}$ )

You're standing at  $0^{\text{th}}$  index & want to reach  $(N-1)^{\text{th}}$  index. For each jump you take a cost of 1 unit.

And from any  $i^{\text{th}}$  digit, you can go to  $i-1$  or  $i+1$

or any other digit  $S_j$  such that  $S_i = S_j$

Find the min jumps.



0 1 2 1 3 4 4 3

worst case → you need to touch  
every digit only  
once

→ 19

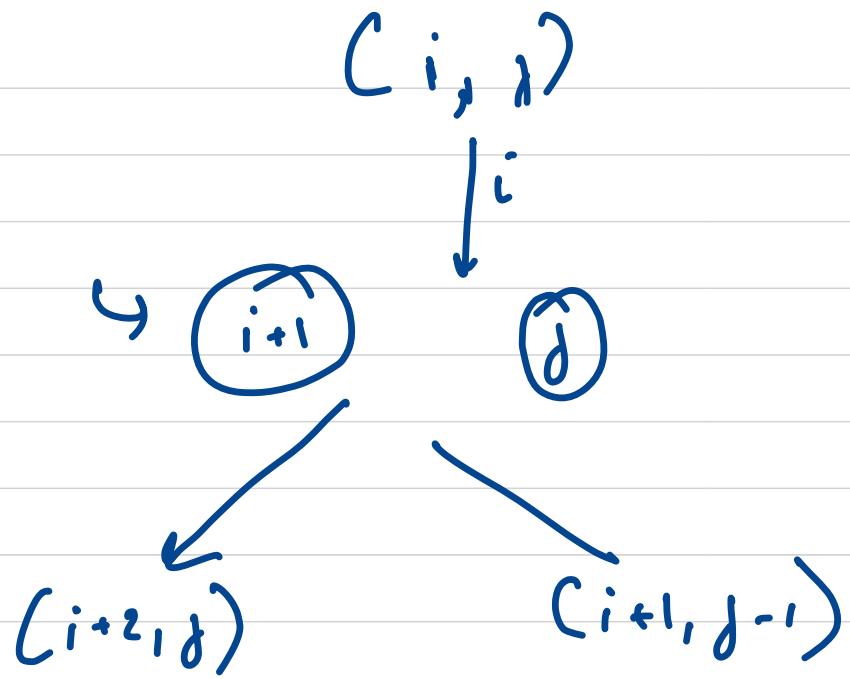
0 0 1 0 0 2 2 2 3 3 4 5 6 6 0 2 3 6 8 9

0 1 2 1 3 4 4 3

0 1 2 3 7 8 6 2

0 → 1 → 2





## Disjoint Set Union

Qn Lets say we want to create clusters

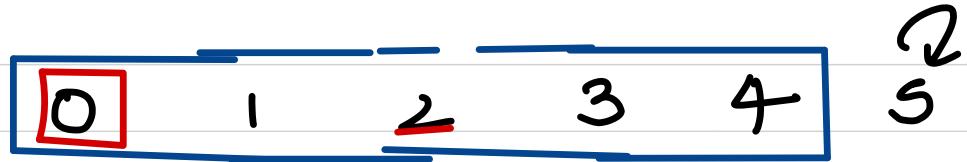
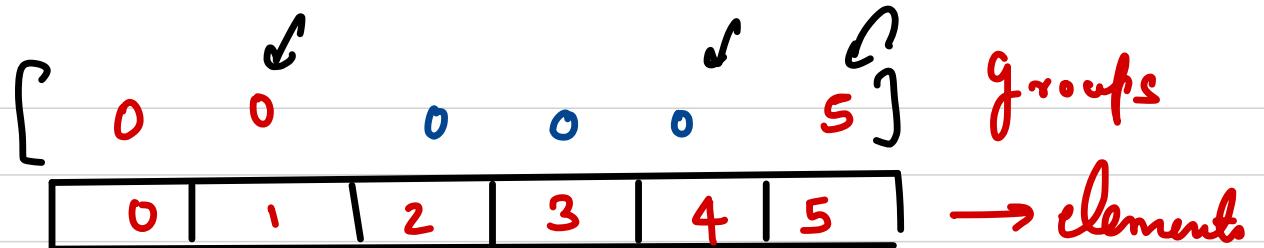
We have a set of elements and we need to group them. Sometimes you might be asked to return the group of any element belongs to.

To uniquely identify a group, we will pick any element of the group & name it the leader/representative / parent of the group.

# union (a,b) → Adds b to the group of a or

vice - versa.

# get(x)/find(x) → To what group the element  
x belongs to.



union(0,1)

union(2,3)

union(3,4)  
union(1,3)

find(1) → 0

find(5) → 5

find(4) → 0

int find (x) {  
 ↗  $O(1)$

return par[x];

}

void union (a,b) {

a = find (a);  
 ↗  $O(n)$

b = find (b);

for i in 1..n {  
 if par[i] == b  
 par[i] = a

,

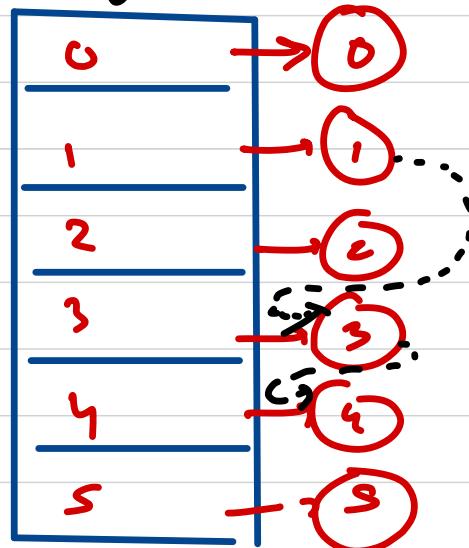
3

Can we  
optimize?

How about storing the grouping mechanism such that we represent it as vector of vector or

vector of ll

head tail



union(1,3)

doubly list

find / get  $\rightarrow O(n)$

union  $\rightarrow \underline{O(1)}$

The core problem is, if we add  $n$  elements  
then the operation of updating parent array  
for each element is  $O(n)$

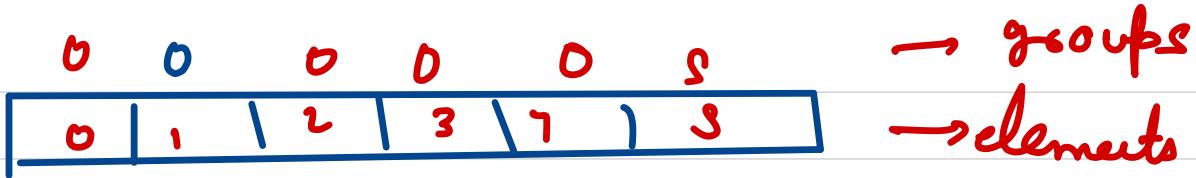
Amortized

$$\frac{1 \times O(n)}{n} \rightarrow \underline{\underline{O(n)}}$$

Somehow optimize updating the parent.

Now let's say we will not union  $b \rightarrow a$  directly

We can maintain sizes of the group, and we will always add smaller group to the bigger one.



union(0,1)

ops → 1

union(2,3)

union(1,0)

union(2,1)

What is the benefit

1

1

:

1

2

2

2

:

2

4

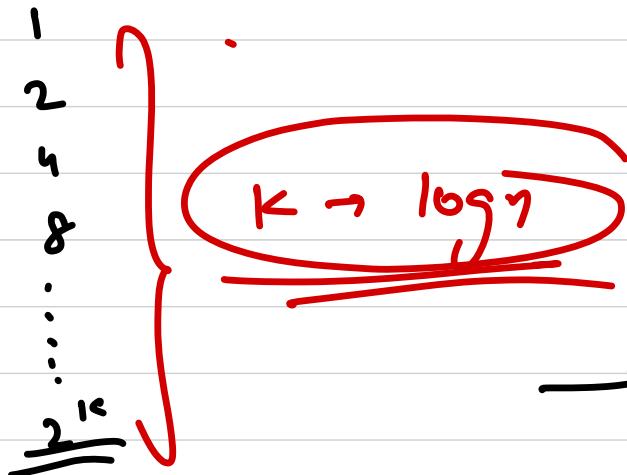
4

4

8

union  
by  
size

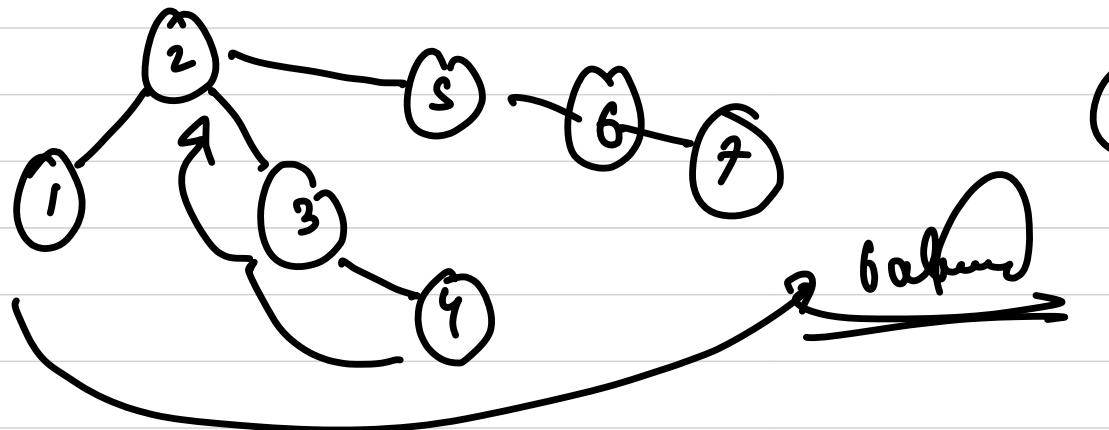
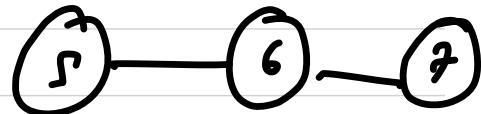
$$\cancel{\frac{1}{2} \times (1 \log n)} \rightarrow \cancel{O(\log n)}$$



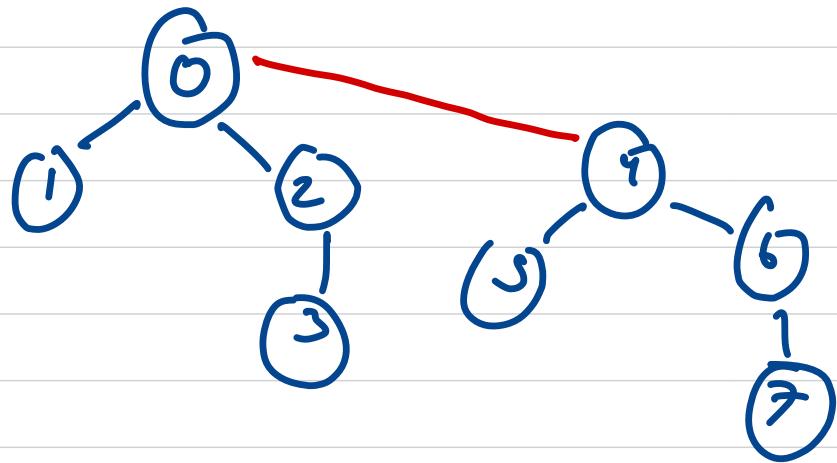
$$1 \times \frac{n}{2} + 2 \times \frac{n}{4} + 4 \times \frac{n}{8} \dots$$

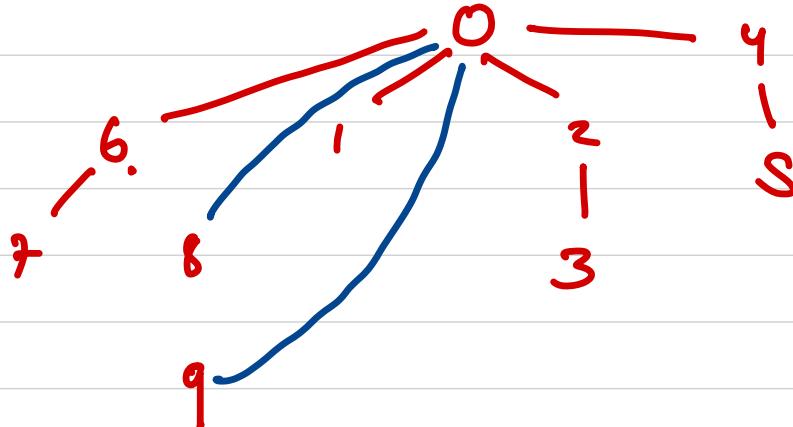
→ union →  $\underline{\log n}$   
get →  $\underline{\log n}$

curve  
obj  
sw



semic





1  
2  
4  
8  
16  
 $\vdots$   
 $2^k$

$\log \gamma$

versus  
alpha

$\log^2 \gamma$

path  
compression

union by  
rank

union by  
rank

$\log^* n \rightarrow$  extremely slow growing func<sup>n</sup>

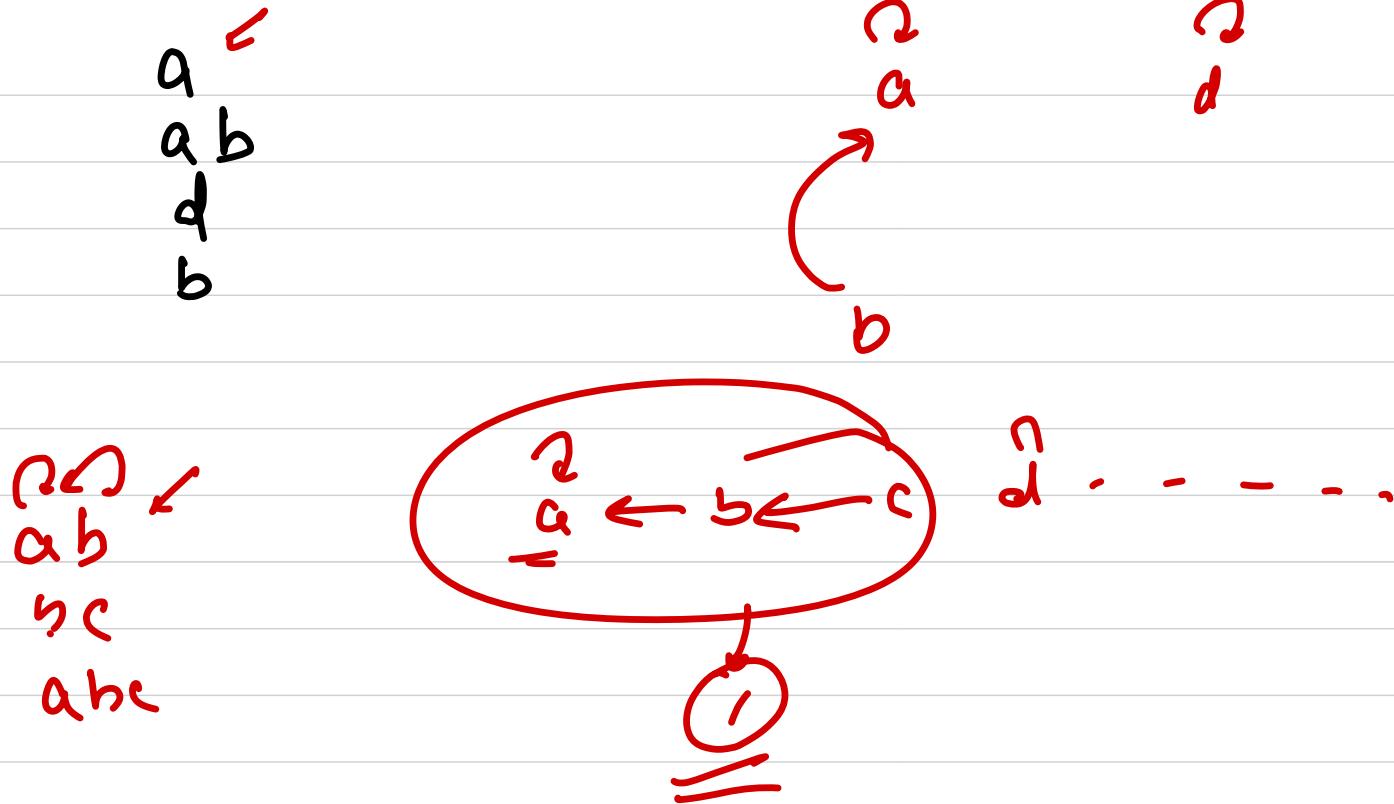
$\log^* n$  no. of steps we need to take

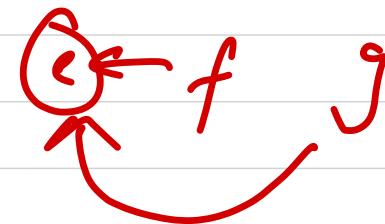
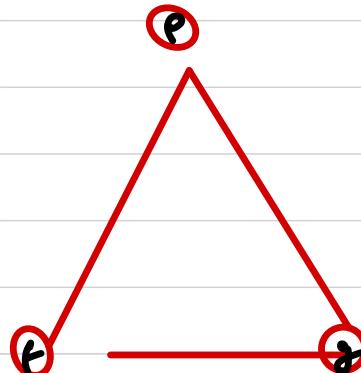
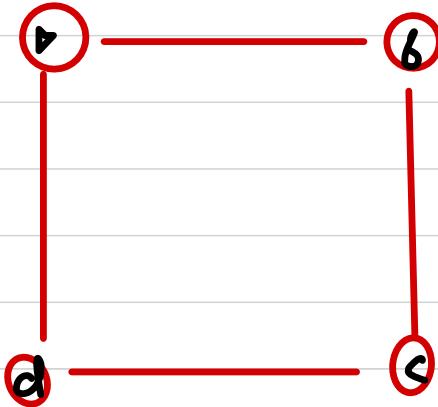
$\log_2$  on the value of  $n$  to make it

smaller than 1.

$$n \xrightarrow{\underline{\underline{2^{16}}}}$$

$$\log 2^{16} \rightarrow \log(16) \rightarrow \log(4) \log(2) \rightarrow 1$$





~~Q3~~ Given,  $n$  persons standing in a row at pos  $\{1-n\}$ . We can do 2 ops -

①  $-x \rightarrow$  remove the person at pos  $x$ .

②  $?x \rightarrow$  find the nearest person to the right of  $x$  that is still present (means not removed)

$$\boxed{n \leq 10^6}$$
$$q \leq 10^6$$

$$n \Rightarrow q = 3$$

$$\begin{array}{r} -2 \\ -3 \\ \hline ?1 \end{array}$$

$$\rightarrow \underline{\underline{1}}$$

~~1 X X 4 5~~



? 1

- 2

{ 1

union(x, y)



- x → union (x, x+1)

? x → get (x+1)

Q: Given an undirected graph, Some ops are given as -

remove a b → remove the edge between a and b vertex

get a b → tell if a & b are in same component.

after all the given no edges will be left,

$$v \leq 5 \times 10^4$$

$$e \leq 10^5$$

$$q \leq 1.5 \times 10^4$$



a — d  
b — c

?  
f  
j

Offline  
query

Answer a d

run d c

run f e

get b c — no

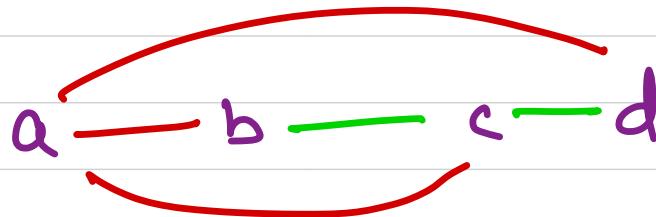
Answer c b  
get b c — no

get a f — no

d. Given a graph with  $v$  nodes, &  $e$  edges , every edge has a color either red or green . Given src & dest , find the shortest path such that path starts from red edge & ends at green edge . & we can switch from a red to green edge only once .

src - a  
dest - d

$v \leq 10^3$   
 $e \leq 10^3$





Do BFS from both sides



first do bfs from src - dest with only red edges

second do bfs from dest - src with only green edges

src

o

b

c

d

des

o

a — s — d

result

~~sd~~