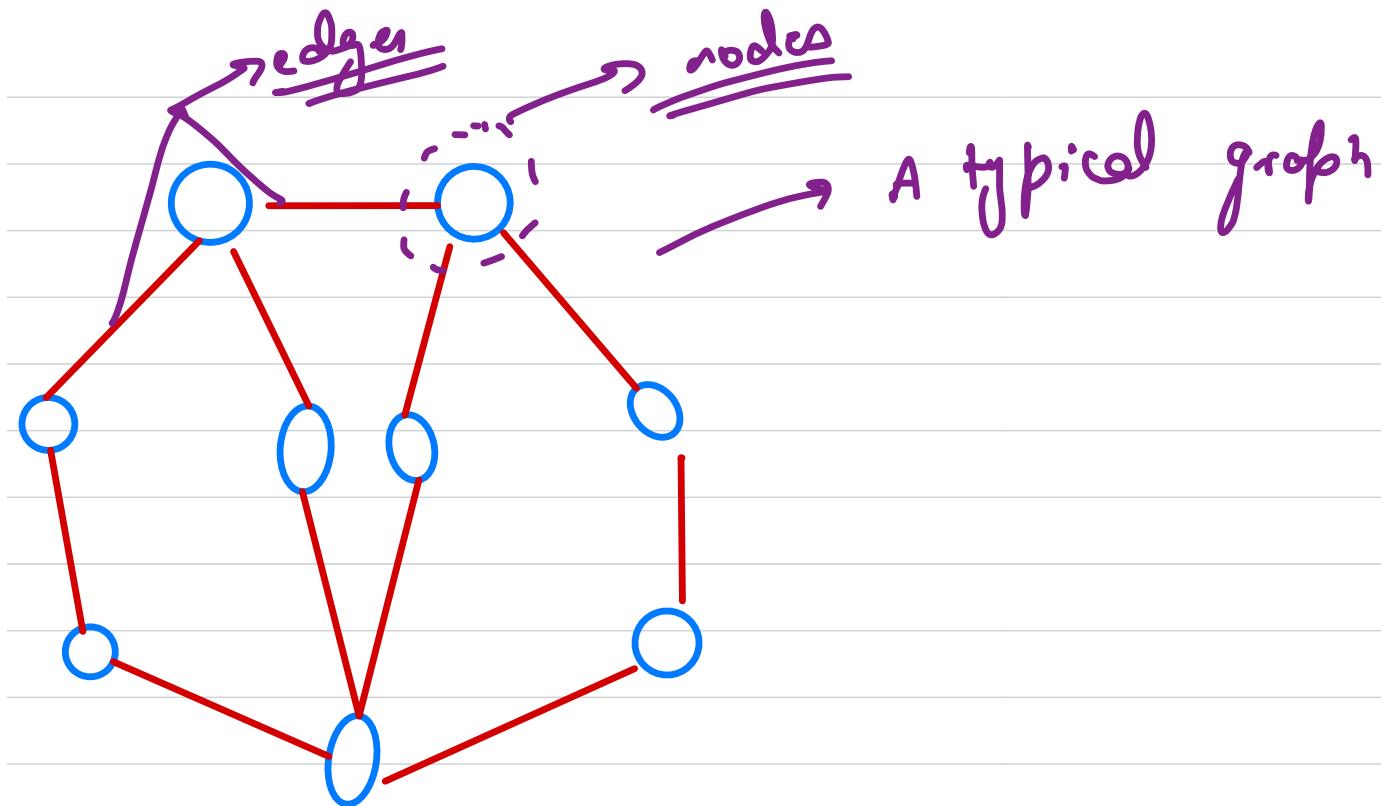


## Graphs

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



Biology       metabolic networks  
                PPI (Protein Protein Interaction)

Electrical    → Circuit Organization

CSE       shortest path  
                Routing algo  
                graph dbs

Uber

Ola

Snappy

zonato

foodpanda



~~Map~~

group  
her  
apply up

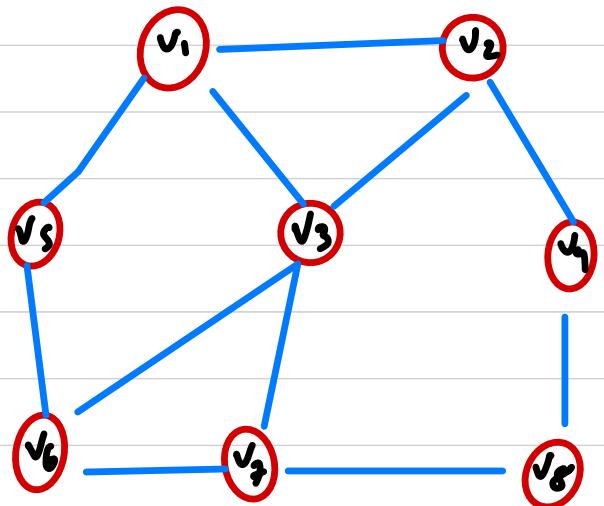
twitter

instagr

fb



formal definition of graph :  $V = \{v_1, v_2, v_3, v_4, \dots, v_e\}$



8 vertices  
11 edges

$E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_8\}, \{v_5, v_6\}, \{v_6, v_7\}, \{v_7, v_8\}\}$

edges  
pairs are  
unordered

$$G = (V, E) \rightarrow$$

mathematical notation of a  
graph

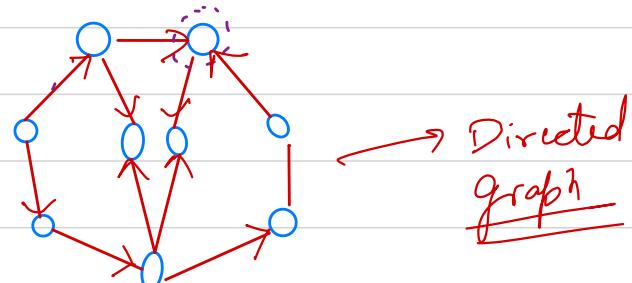
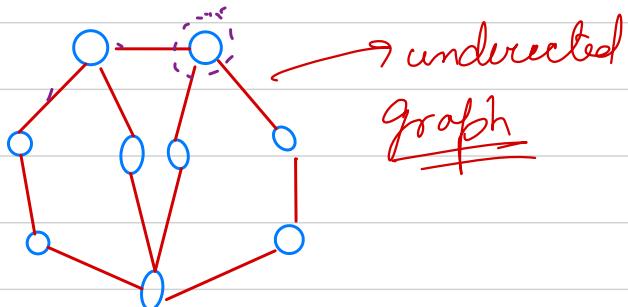
A graph  $G$  is an ordered pair of a set  $V$  vertices &  $E$ , a set of edges.

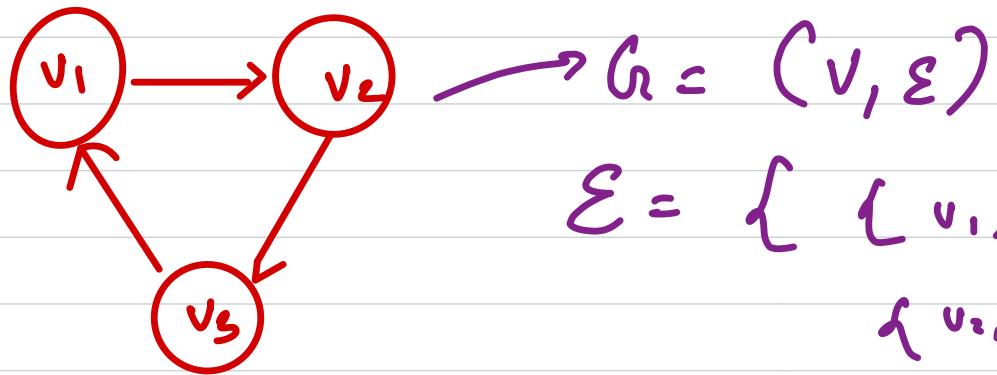
Type of graph :



① Undirected  $\rightarrow$  Facebook

② Directed  $\rightarrow$  Instagram, Twitter





$$G = (V, E)$$

$$E = \{ \{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_1\} \}$$

Set of ordered pairs

Based on edge property

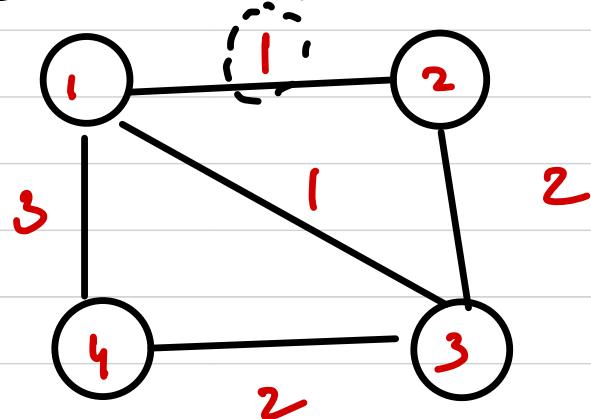
① Weighted

Weighted

② Unweighted

Unweighted

edge weight



Based on no. of edges

(1) Sparse

(2) Dense

## Graph data structure →

↪ It is a non-linear data structure,

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

# Representation of graphs : ↵

(1) Adjacency Matrix

(2) Adjacency List —————> Adjacency Map

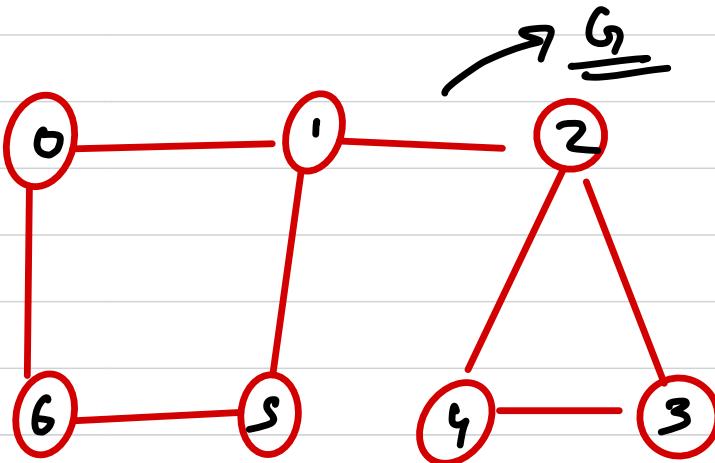
(3) Incidence Matrix

→ edge list

# # Adjacency Matrix

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

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



$\underline{(V \times V) \rightarrow \text{dense}}$

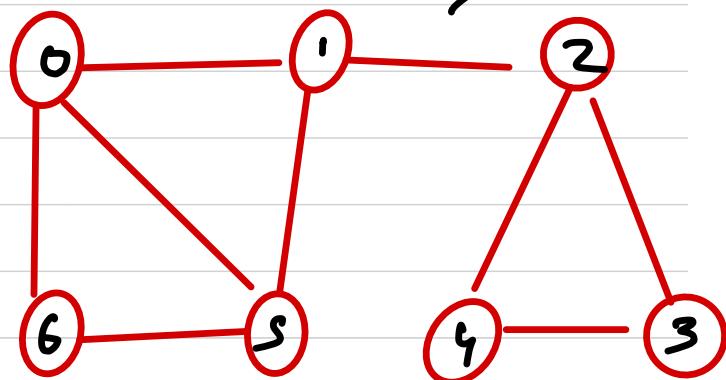
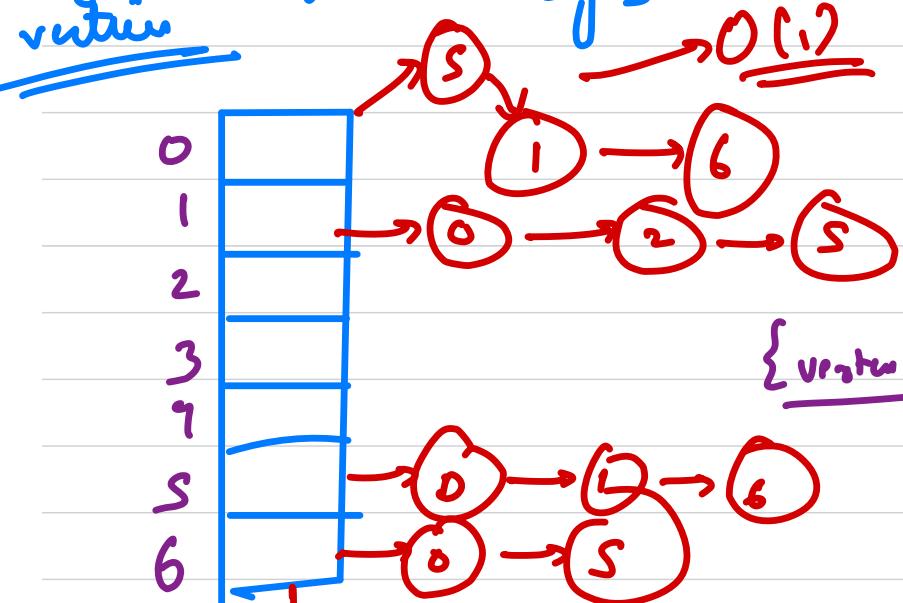
$\underline{|V|} \rightarrow \text{no of vertices}$

$V \times V$

# Adjacency list

added at head  
good for sparse graphs

Array of LL → space optim.



{vertex, wt} → node

$$(V + \alpha E) \approx O(V+E)$$

# adjacency map

↓  
array of hashmap

hasEdge (v, v) →  $\Theta(1)$

# # Incidence Matrix

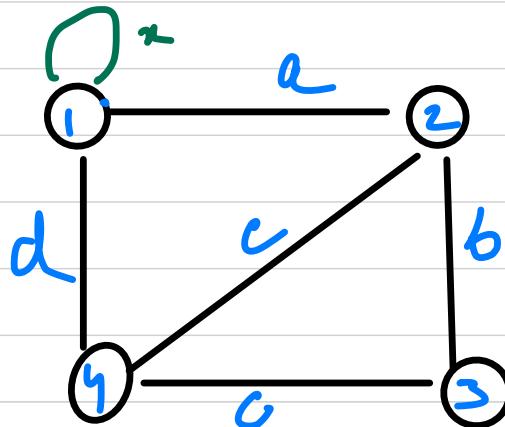
$$V = \{1, 2, 3, 4\}$$

$$E = \{a, b, c, d, e\}$$

$M =$

	a	b	c	d	e	
1	1	0	0	1	0	x
2	1	1	0	0	1	z
3	0	1	1	0	0	y
4	0	0	1	1	1	w

rows  $\rightarrow$  vertices  
 $(V \times E)$



$(V \times E)$  dimension

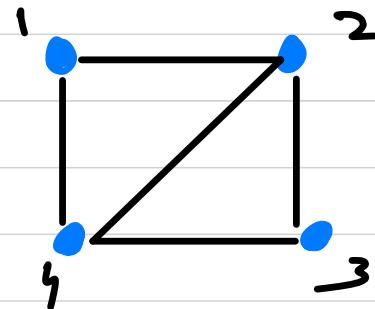
$$M_{ij} = \begin{cases} 1 & \text{if the } i^{\text{th}} \text{ vertex} \\ & \text{belongs to the } j^{\text{th}} \text{ edge.} \\ 0 & \text{else} \end{cases}$$

$$\text{rows-Sum} = \underline{d \quad r \quad v}$$

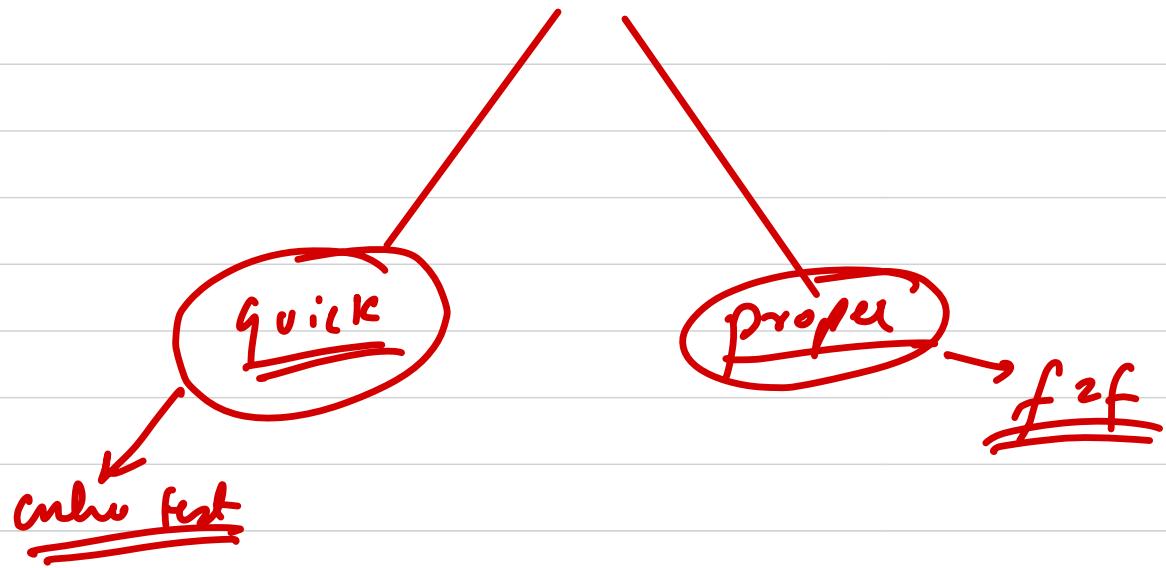
$$\text{column-Sum} = \underline{\alpha}$$

~~Q~~ What is a degree ??

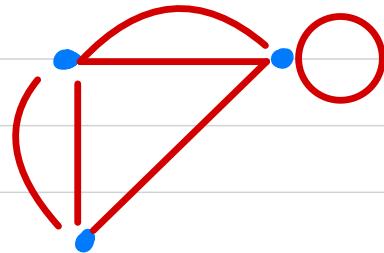
Degree of a vertex in a graph G, is the total no. of edges incident to it / associated with it  
(undirected graphs)



# note → In a directed graph, the outdegree of a vertex is total no. of outgoing edges & indegree is total no. of incoming edges.



# Multi graph  $\rightarrow$  an undirected graph with multiple edges and loops allowed.



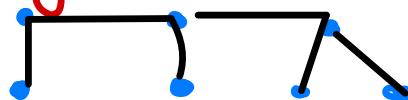
# Simple Graph  $\rightarrow$  An undirected graph in which both multiple edges & loops are not allowed.

Complete Graph  $\rightarrow$  A graph in which every vertex is directly connected to every other vertex.

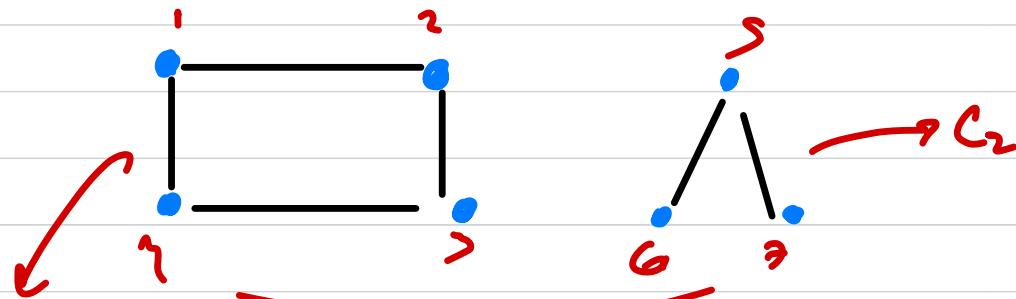


complete graph

Connected graph  $\rightarrow$  A connected graph has a path from any vertex to other vertex, not necessarily direct.



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



$$C_1 = \left( \{1, 2, 3, 4, 5, 6, 7\}, \{ \{1, 2\} \{2, 3\} \{3, 4\} \{4, 5\} \{5, 6\} \{6, 7\} \{7, 1\} \{1, 2\} \{2, 3\} \{3, 4\} \{4, 5\} \{5, 6\} \{6, 7\} \} \right)$$

# 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



$$V = \{v_1, v_2, v_3, v_4\}$$

Such that edge set of a graph is

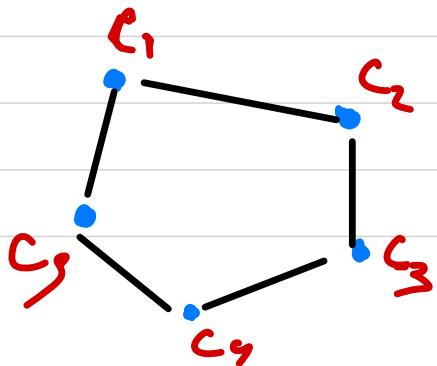
$$E = \{v_i v_{i+1} \mid i \in [1, n-1]\}$$

~~Cycle~~ → A cycle  $C_n$  is a graph whose vertices can be arranged in a cycle sequence

$$V = \{v_1, v_2, v_3, v_4, \dots, v_n\} \text{ such}$$

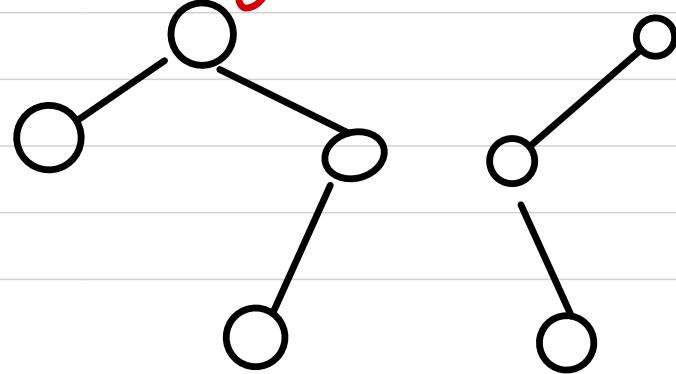
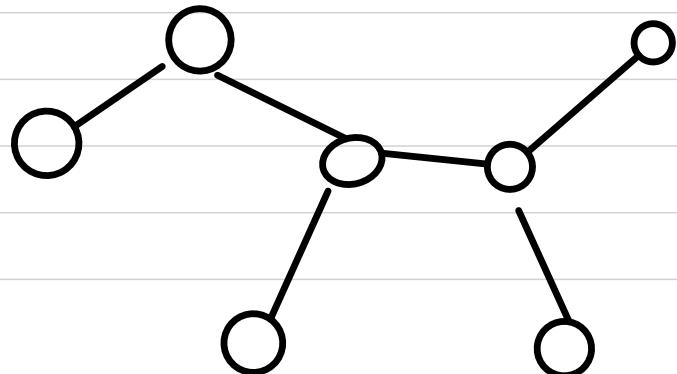
that edge set is

$$E = \{v_i, v_{i+1} \mid i \in [1, n-1]\} \cup \{v_1, v_n\}$$

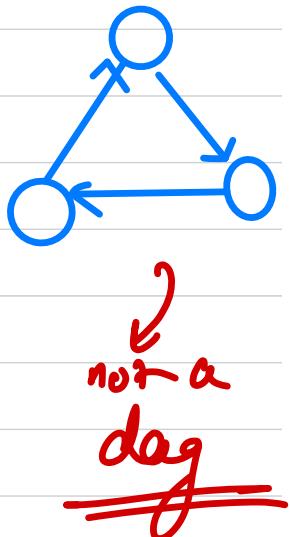
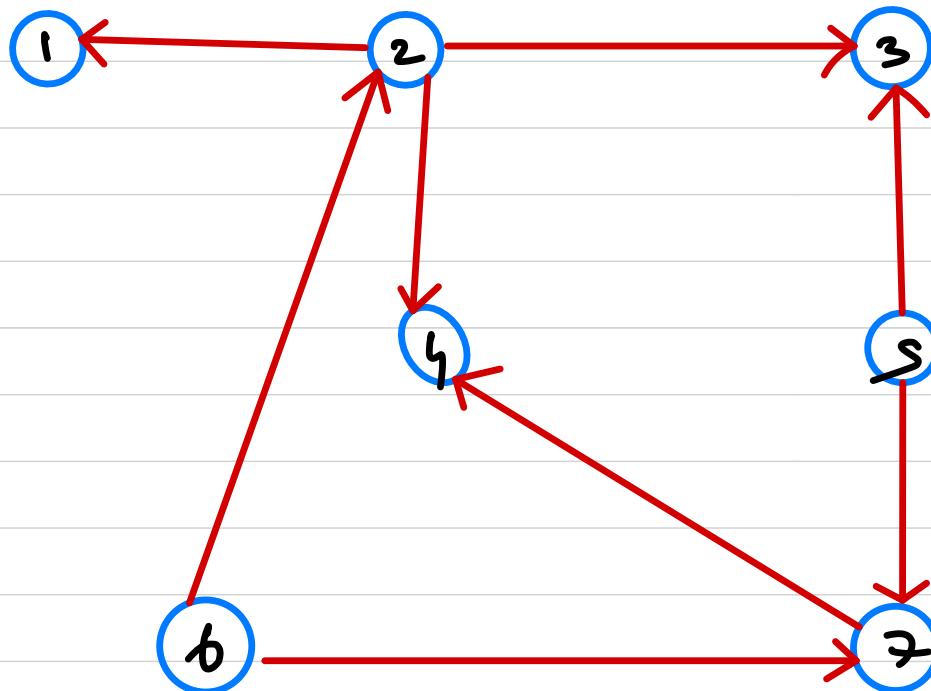


TREE → Tree is a connected graph with no cycles. =

forest → If we remove an edge from tree, we get a forest viz collection of trees.



# DAG (Directed acyclic graph)



# How to Read Graphs

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

## Graph Traversals

- ① Depth first search
- ② Breadth first search

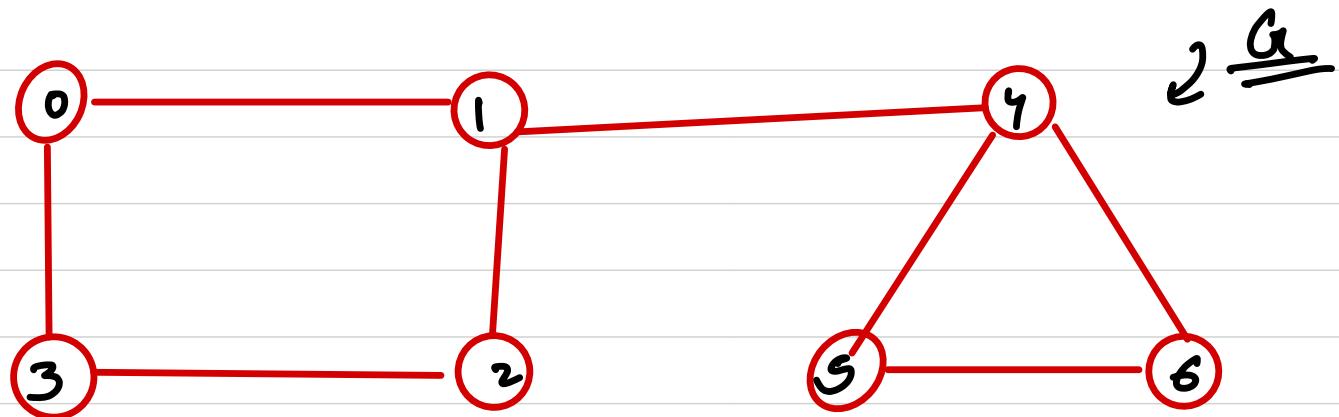
# Depth first Search

Motivation problem:

Given a graph calculate all paths between  
2 vertices

OR

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



is there a path from 0 to 5 i.e.?

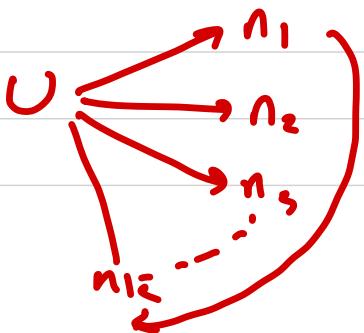
What is the most easiest query for this i.e.?

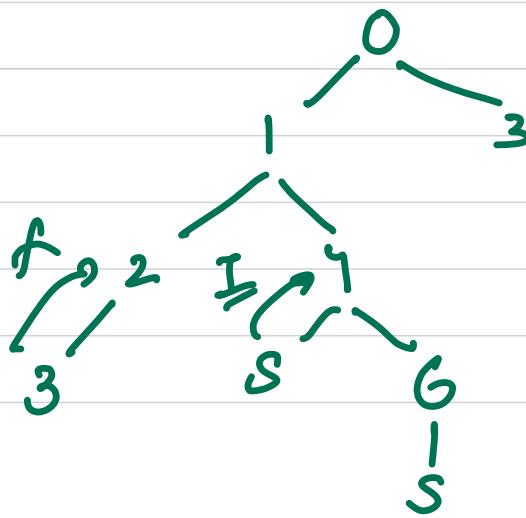
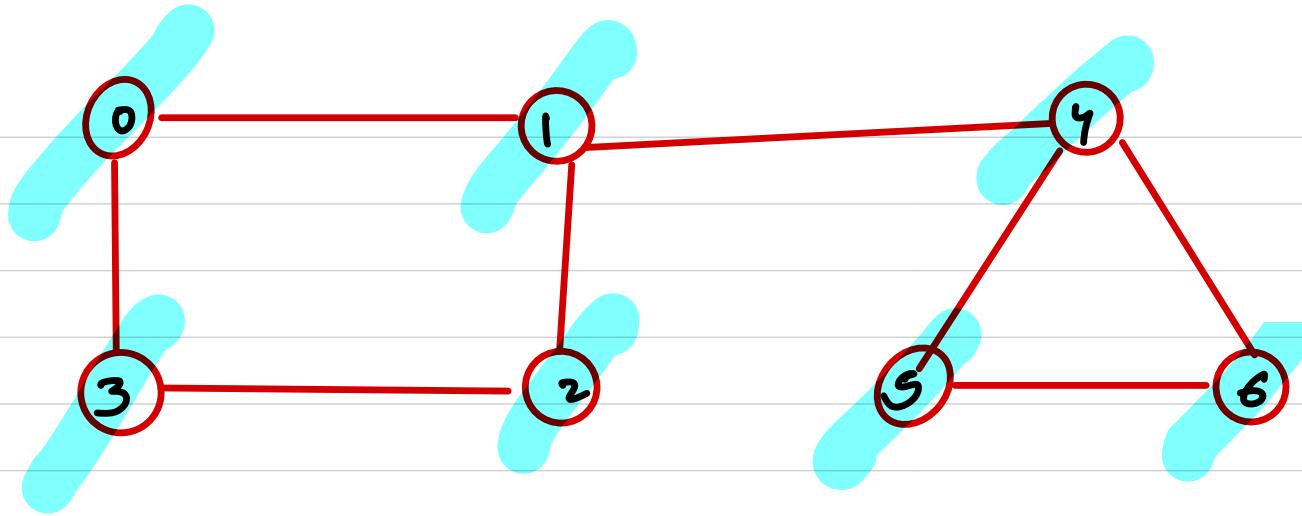
→ neighbours

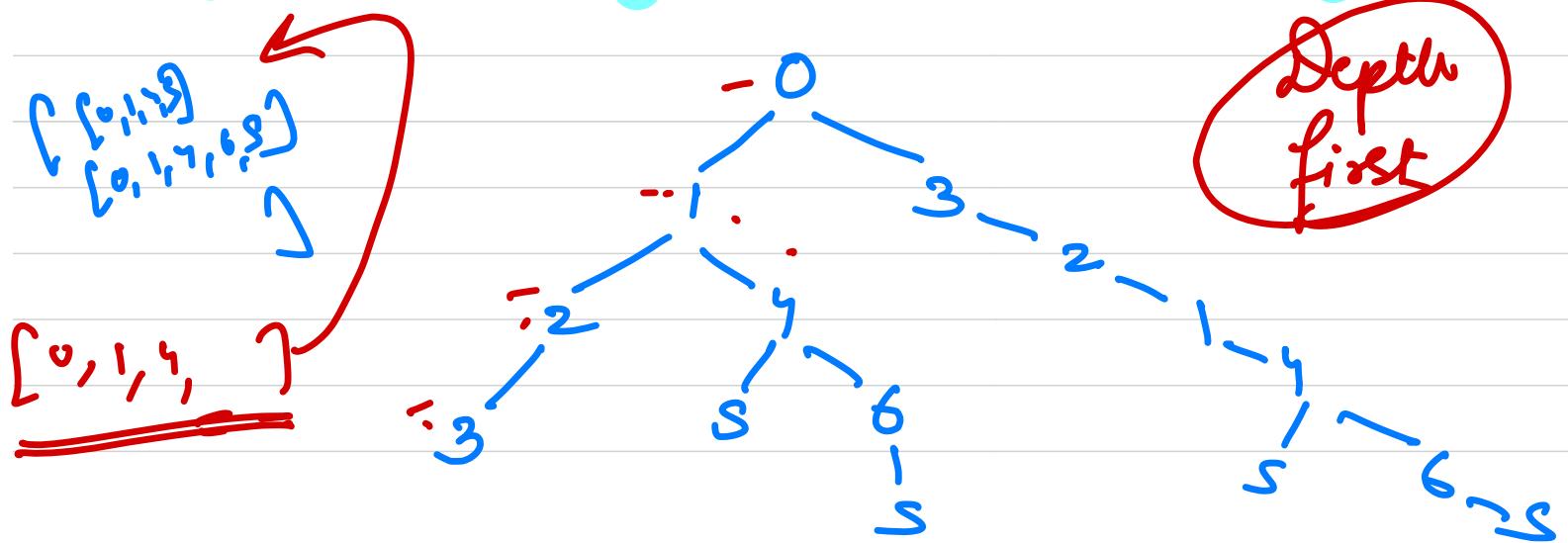
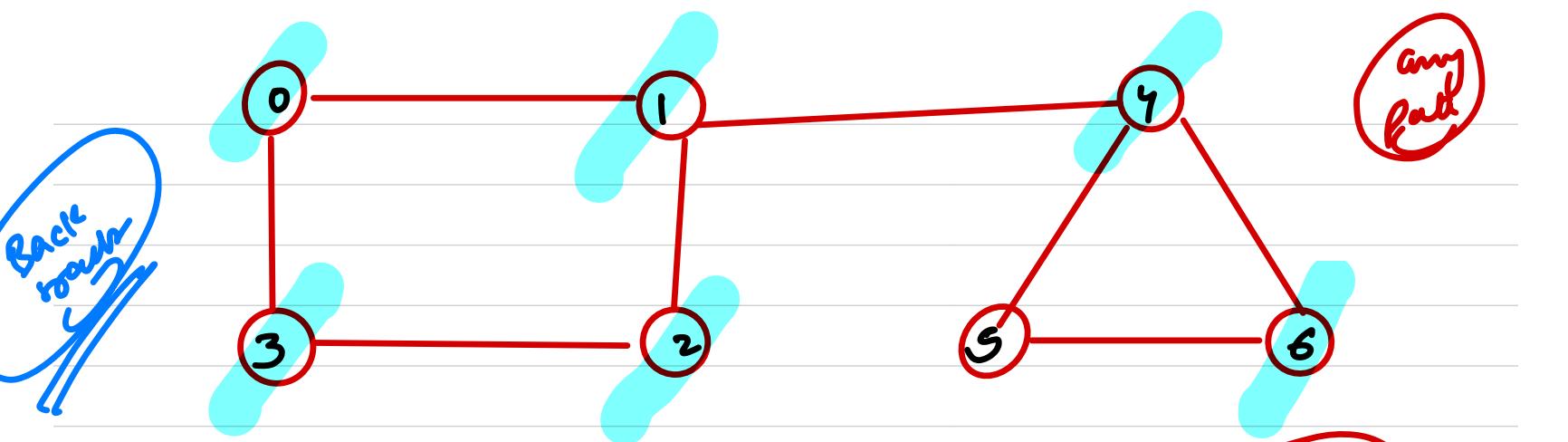
path always count for immediate neighbours

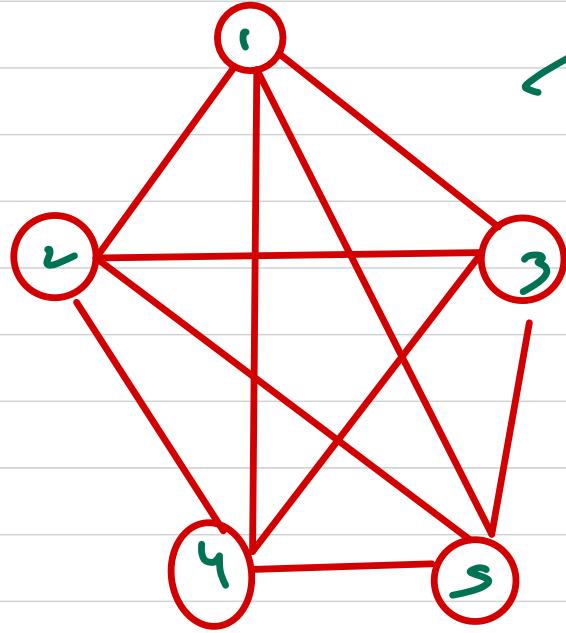
$$f(u, v) = \begin{cases} f(n_1, v) \\ \text{or} \\ f(n_2, v) \\ \text{or} \\ f(n_3, v) \\ \vdots \\ f(n_k, v) \end{cases}$$

whether there is  
a path from  
u to v.







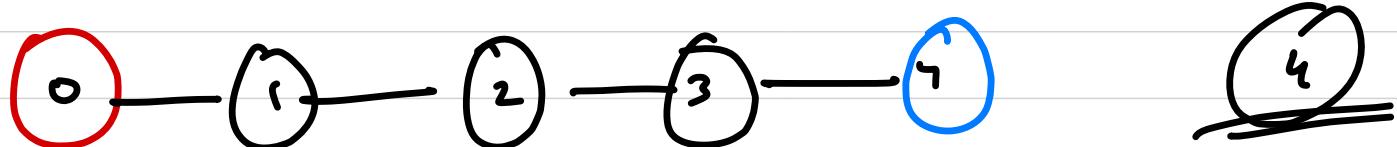
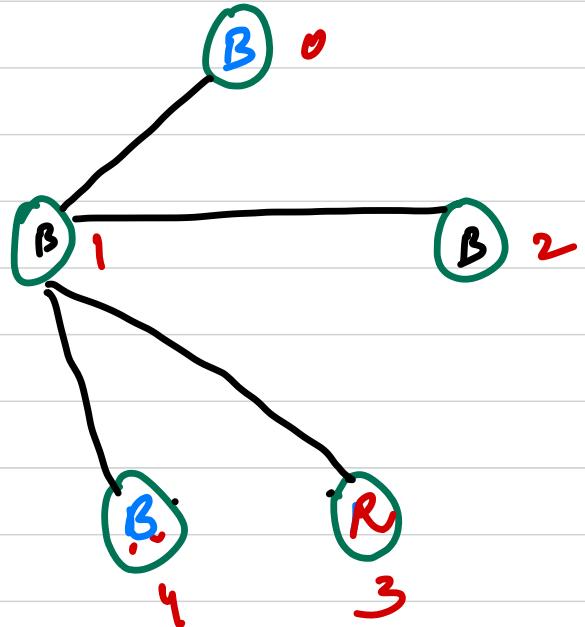


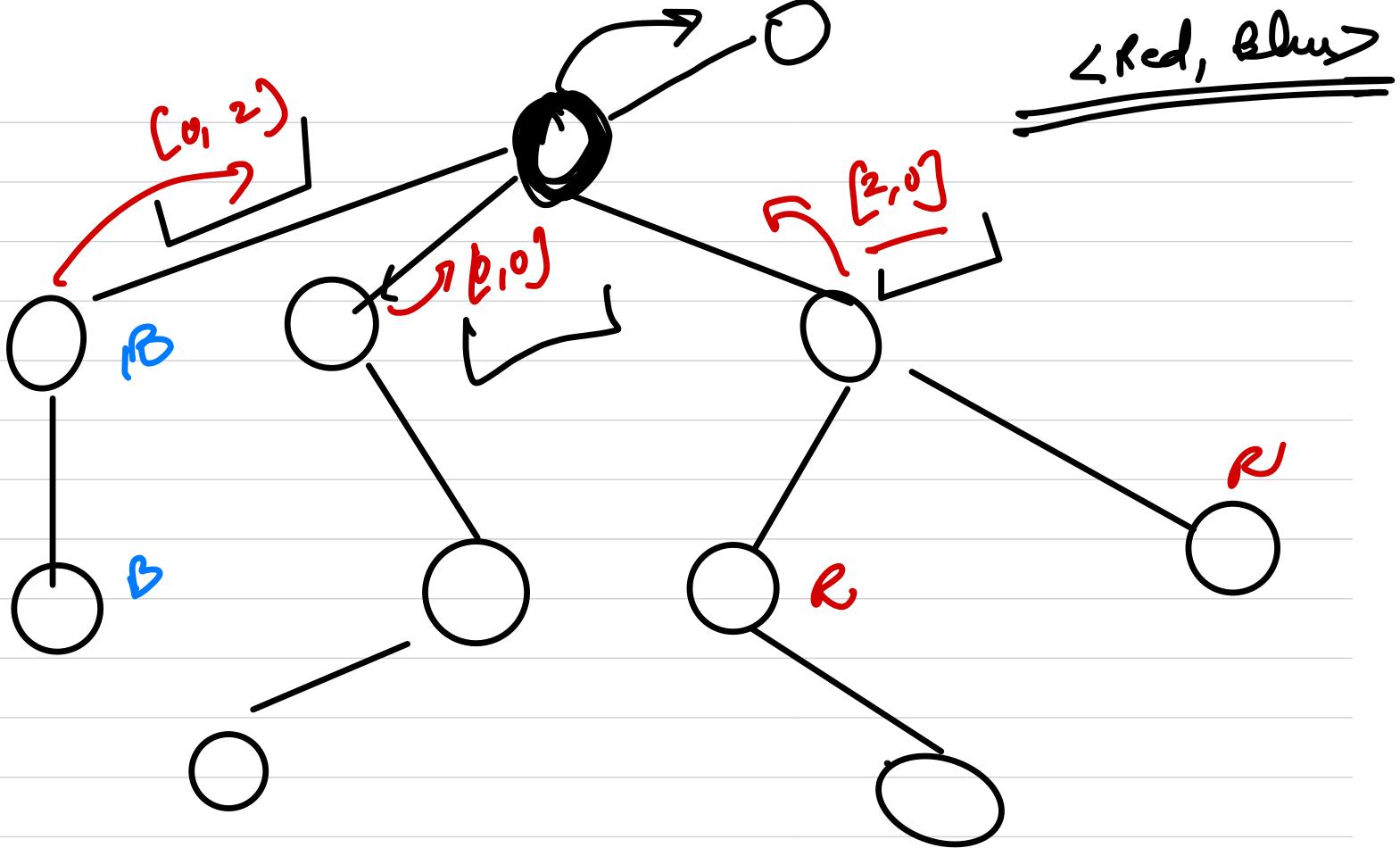
complete graph

1, 2, 3

$O(v!)$

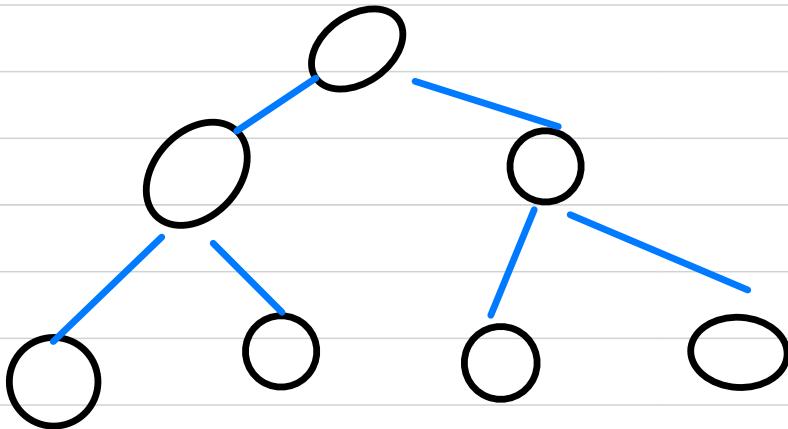
~~$\phi = 1$~~  You are given an undirected tree with  $v$  vertices. The vertices can be of any one of the following colors : Red, Blue, Black. Tree contains atleast one red and one blue node. You can remove an edge such that we get 2 trees where none of the tree has both red and blue color nodes together. Find how many such edges we can remove.  $v \leq 3 \times 10^3$





True  $\rightarrow$  V nodes

edges ??



Prove that a tree with  $n$  nodes has  $n-1$

edges.

Pm1  $\rightarrow$  assume  $f(n) \rightarrow n-1$

no. of edges for  
 $n$  nodes.

$$\begin{aligned} f(1) &\rightarrow 0 \\ f(2) &\rightarrow 1 \end{aligned}$$

Base Case

assume  $f(n) \rightarrow \underline{\text{base}}$

To procure  $\underline{f(n+1)}$

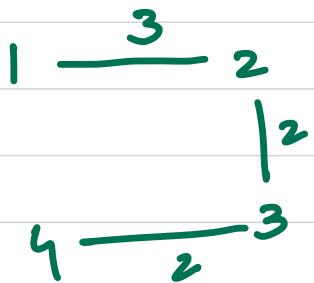
no. of edges will be  $(n-1)$  + no. of edges required for  $(n+1)^{\text{th}}$  node.

Every node that will be added to a tree needs one edge.

$$\begin{aligned} \underline{f(n+1)} &= f(n) + 1 \\ &= f(n-1) + 1 \Rightarrow (\underline{n+1}) - 1 \\ f(n) &= \underline{n-1} \quad \underline{\text{H.P.}} \end{aligned}$$

Holi

(Spoj)



$$\begin{aligned}1 &\rightarrow 3 + 2 \rightarrow 5 \\3 &\rightarrow 2 + 3 \rightarrow 5 \\4 &\rightarrow 2 + 2 \approx 4 \\2 &\rightarrow 2 + 2 \rightarrow 7\end{aligned}$$

1 ⊂ 2

$$1 \rightarrow 3$$

1 ⊂ 3

$$2 \rightarrow 3$$

$$\begin{matrix}4 - 2 \\3 - 2\end{matrix}$$

10

18 ← solutio

# Brute force

1. 2. 3..... N

→ We can have  $N!$

permutations

1 2 3 4

1 4 2 3

$N$  nodes



$(N-1)$  edges

$2 - 9$



Tree (Weighted)

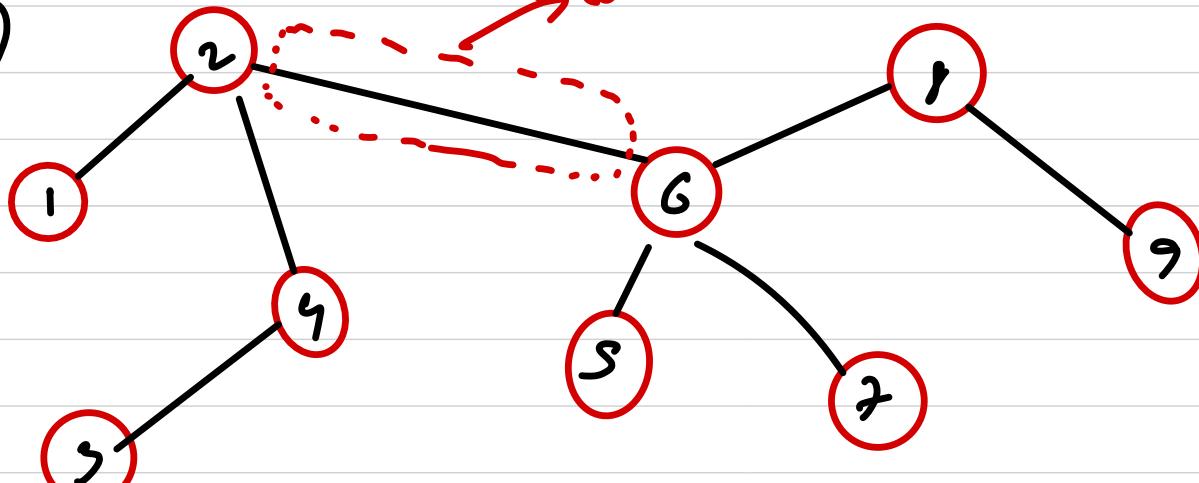
max combination of each edge

$\Sigma^k O(v \times \epsilon)$

$\sqrt{\epsilon} + \epsilon^2$

$O(CV^2)$

$1 - \epsilon$

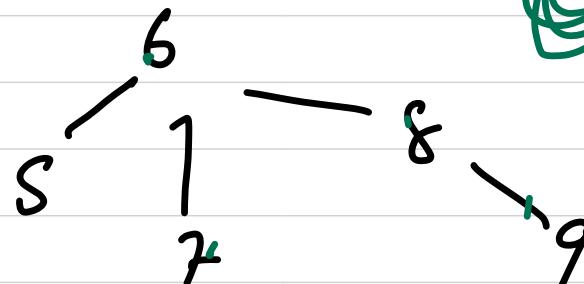
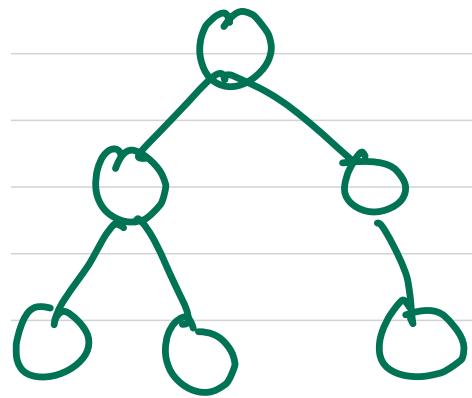
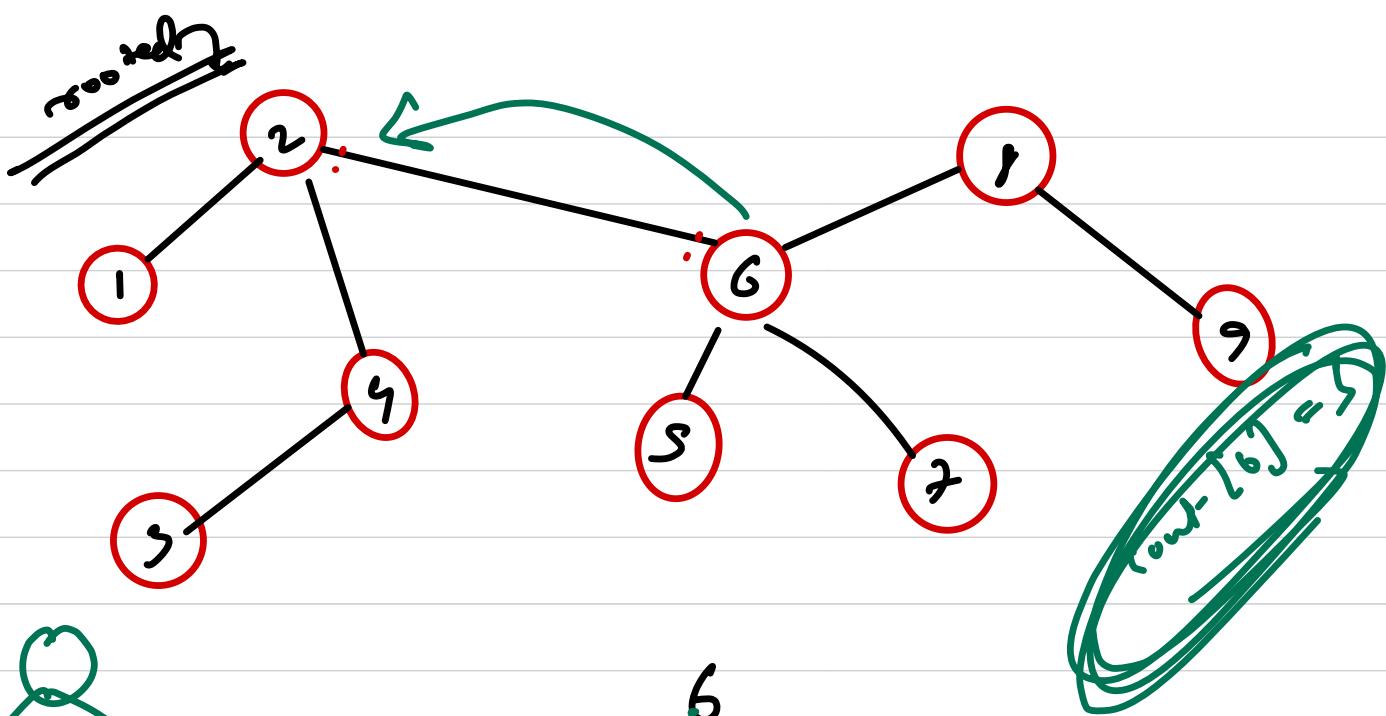


$1 - S$   
 $2 - 6$

$S - 2$   
 $6 - 3$  ---

For any edge  $e$ ,

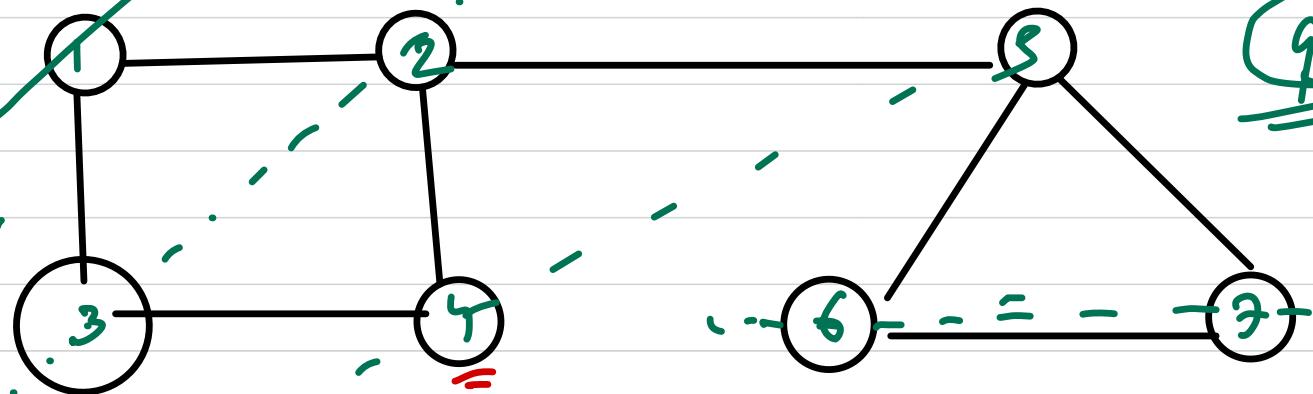
$$\text{Contribution} = 2 \times w + x \min(s_2 - c_1, s_2 - c_2)$$



vis

Breadth first Search

FIFO  
Queue



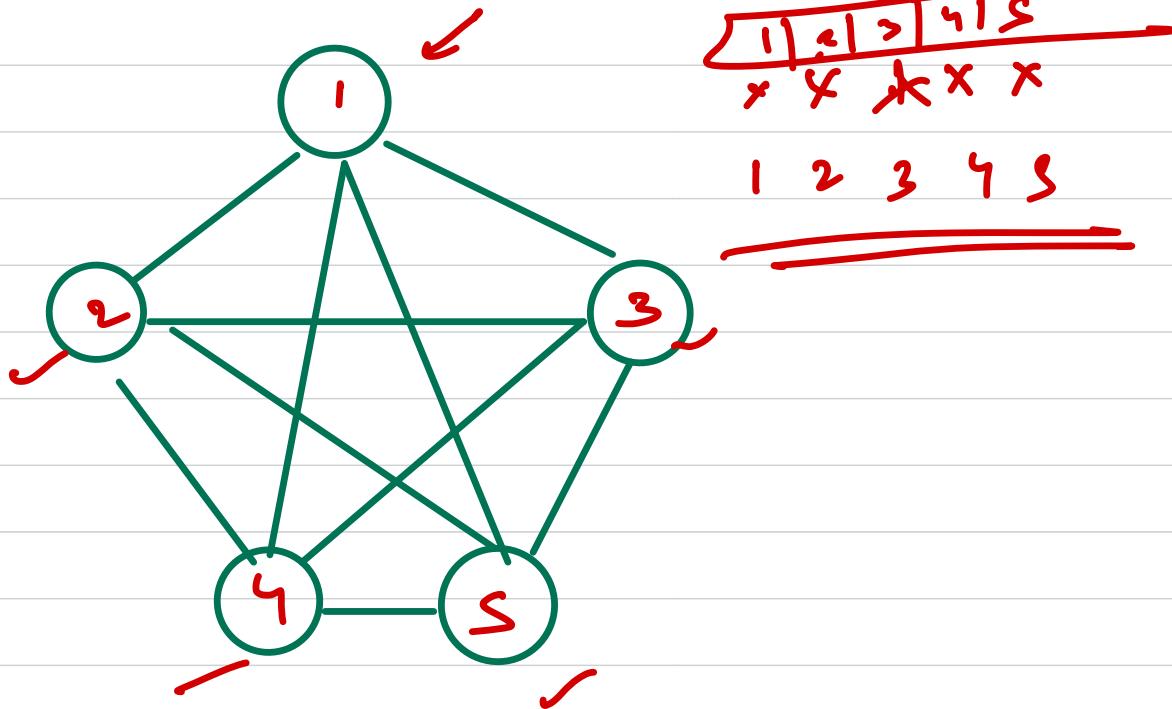
queuing

$1 \times 2, 3, 4, 5, 6, 7$

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

level order traversal

$T C \rightarrow O(V+E)$

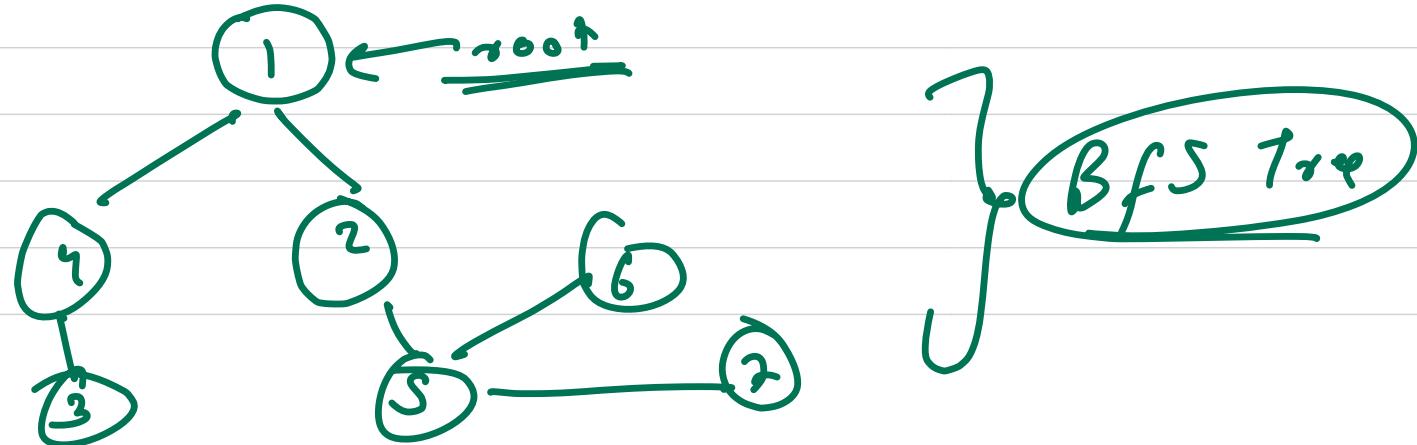
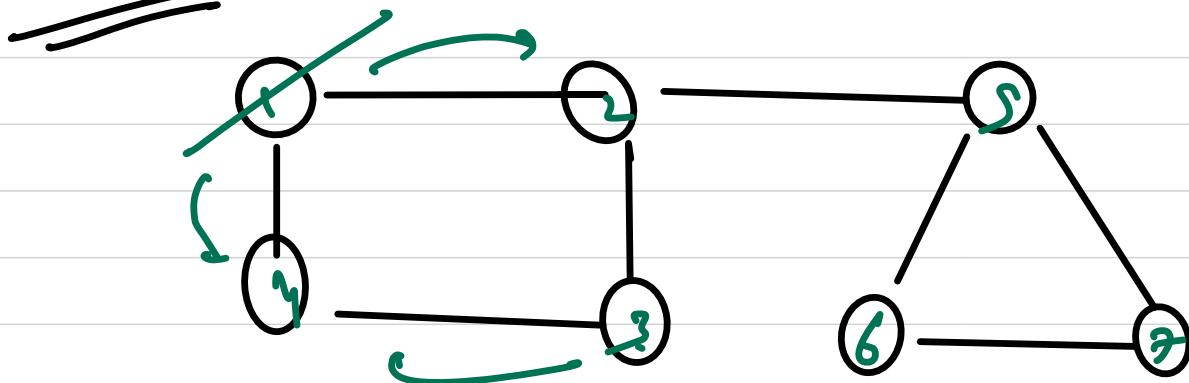


1 2 3 4 5  
x x x x x  
1 2 3 4 5

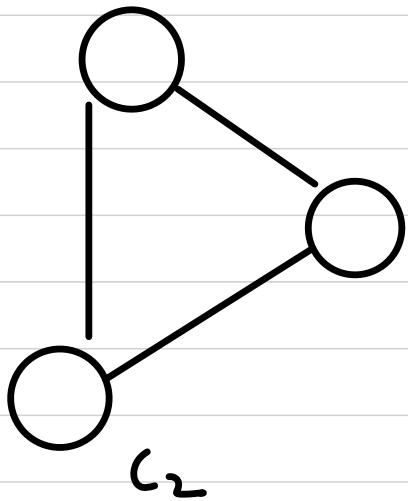
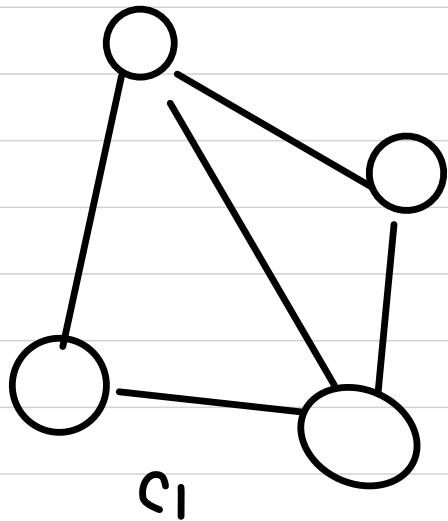
BFS Tree

1, 2, 3, 4, 5

parent L[i][j]



# \* Connected Components

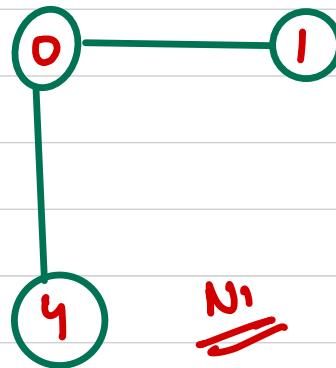


$G$

~~DFS~~ Journey to the moon  $\frac{^n C_2}{\equiv}$   ~~${}^0 C_2$~~   $\rightarrow$  Total

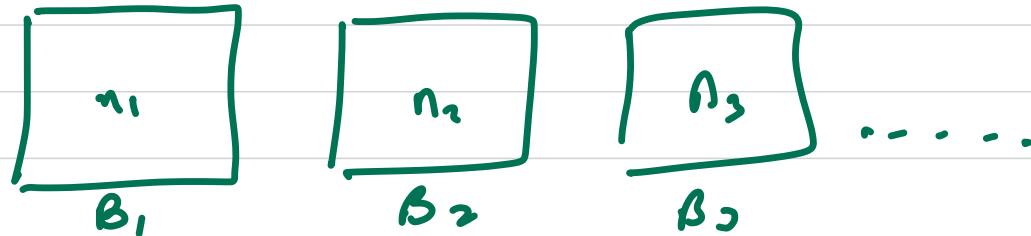
Total -  $\Sigma$  pairs of same nature

~~DFS~~



$\frac{N_o}{\equiv}$

# of lines of code  
~~dfs = # of CC~~



~~Q =~~ You have a permutation of  $N$  numbers  $1, 2, 3, \dots, N$ .

You want to rearrange this permutation to some other permutation  $P$ . You can swap only  $M$  given pairs, whatever no. of times you want. figure

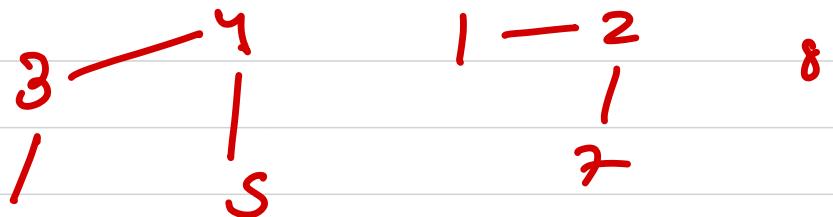
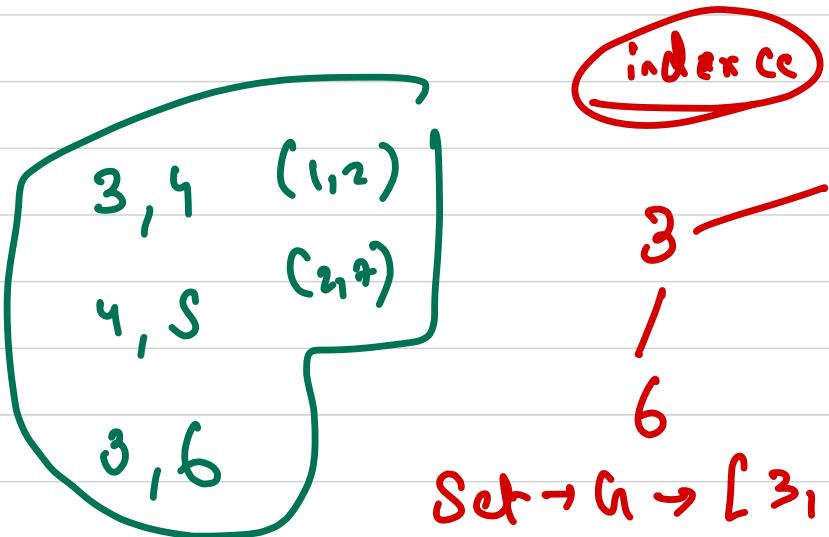
Out if this is possible or not? ?

$$G \rightarrow 1, 3, 2, 4 \quad (3, 4)$$

$$f \rightarrow 1, 4, 2, 3$$

↗ Yes

$\rightarrow$  connected component  $\leftarrow$  direct applicati



$G \rightarrow 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8$

$\rho \rightarrow 7 \ 1 \ 4 \ 6 \ 5 \ 3 \ 2 \ 8 \ \checkmark$

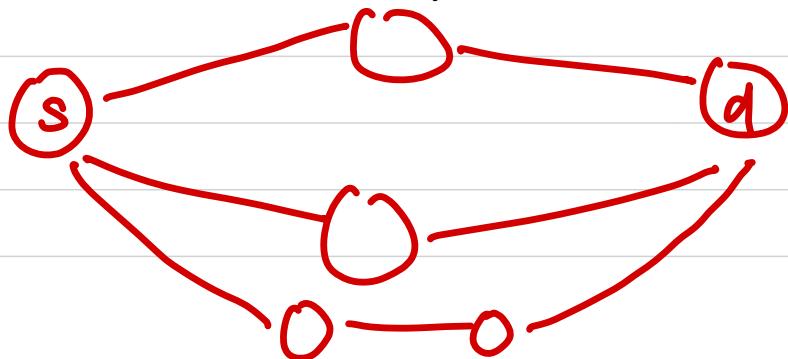
all the elements in one connected component

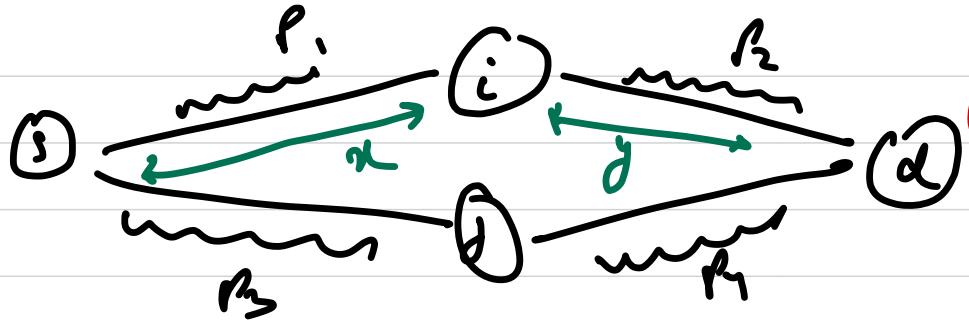
can be arranged in any permutation under

some swaps

$\rightarrow$  ~~BFS~~  $\rightarrow$  ~~graph~~  $\rightarrow$  shortest path

~~O~~ Given a graph, give src & dest nodes,  
find all the nodes which are part of  
at least 1 shortest path.





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

$$\cancel{O(V+E)}$$

$$\cancel{\gamma = d - \alpha}$$

$$\cancel{\alpha - \gamma = d}$$

## Disjoint Set Union

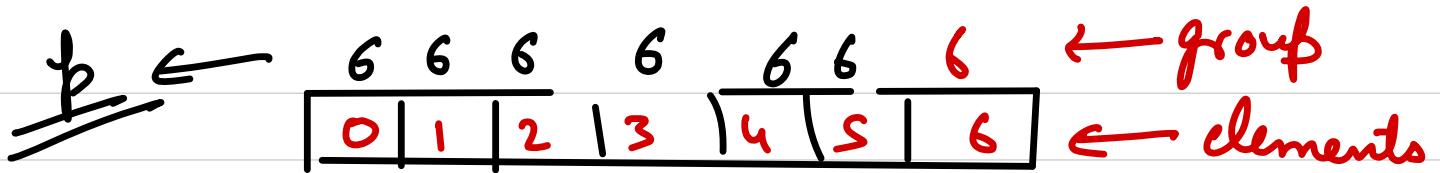
↳ We want to create clusters //

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

↳ To uniquely identify a group, we will pick any one element of the group & name it leader/ parent of group. This parent is the identifier.

1)  $\text{union}(a, b) \rightarrow$  adds  $b$  to the group of  $a$  or  
vice-a-versa.

2)  $\text{get}(x) / \text{find}(x) \rightarrow$  to what group / cluster  
 $x$  belongs  $\equiv$ .



union(0,1)

union(2,3)

union(2,4)

union(2,5)

union(2,0)

union(6,3)

int get (int x) { // O(1)

return p[x];

}

void union (int a, int b) { // O(n)

a = get(a);

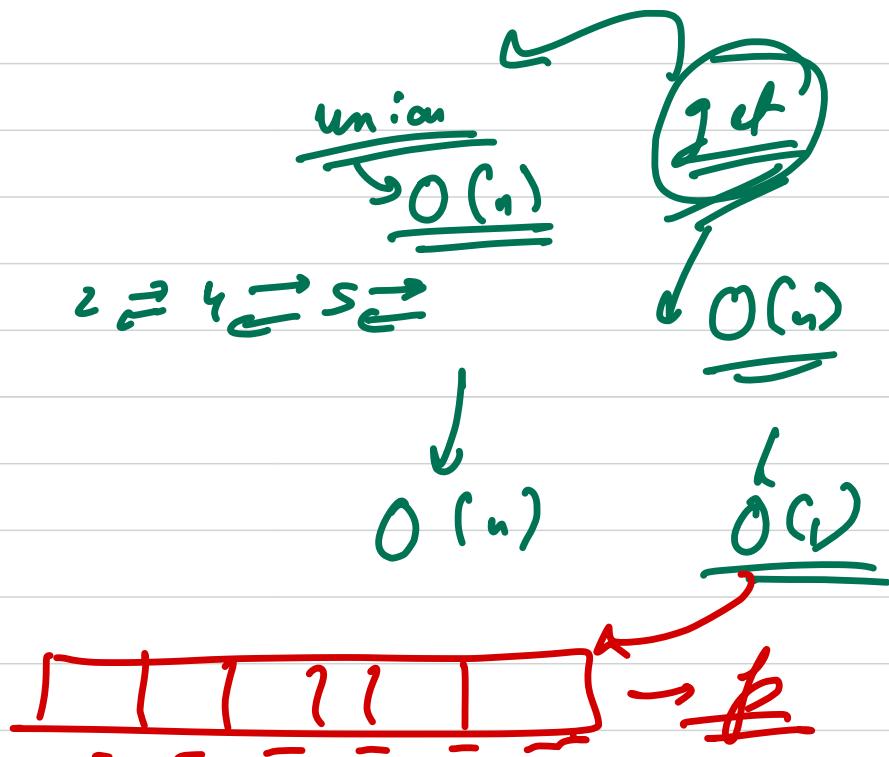
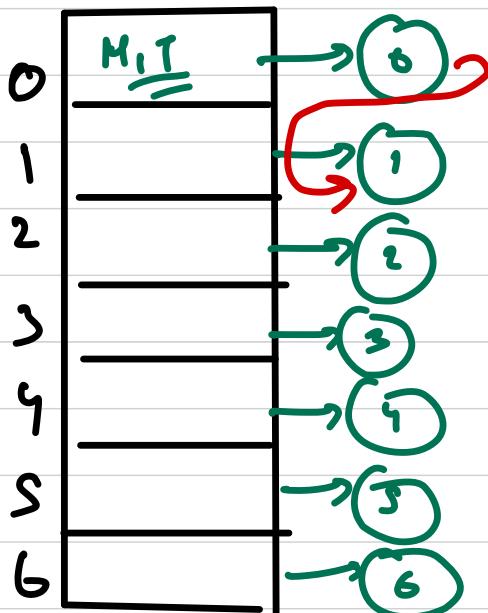
b = get(b);

for (int i = 1; i < n; i++)  
if p[i] == b  
p[i] = a;

}

}

// How about storing groups as ll. / dep



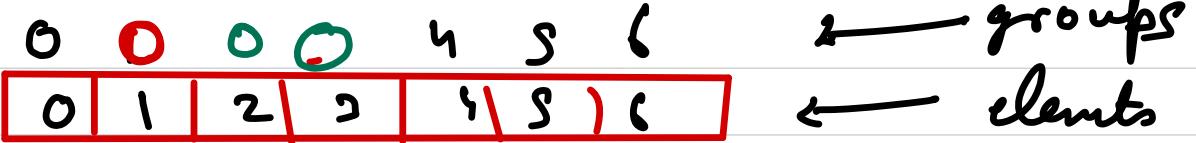
// So what we can see , if we add  $n$  elements then the operation of updating parent array for each element is  $\underline{\mathcal{O}(n)}$

amortized

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

We will not arbitrarily add  $b$  to  $a$  instead  
we can maintain group sizes. And we will  
always add smaller group into bigger group

union  
by  
size



union (0,1)

union (2,3)

union (2,0)

1 - - - 1 → 2 - - 2 - - 4 - -

1  
2  
3  
4  
8

1 . .  
2 . .  
4 . .  
8 . .  
k → g^n

8 ≈ 1

~~connection  
array~~

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

$$\frac{1}{2^k} = 1$$

$\text{C} = \log n$



$$\frac{n}{2} \times C + \frac{n}{4} \times \cancel{\frac{1}{2}C} + \frac{n}{8} \times \cancel{\frac{1}{4}C} + \dots + 1 \times \frac{1}{2^k} C$$

$$n \rightarrow \cancel{k} \times \frac{1}{2} \times \log n$$

$$\frac{1}{2} \rightarrow C \left( \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \dots + \frac{1}{2} \right)$$

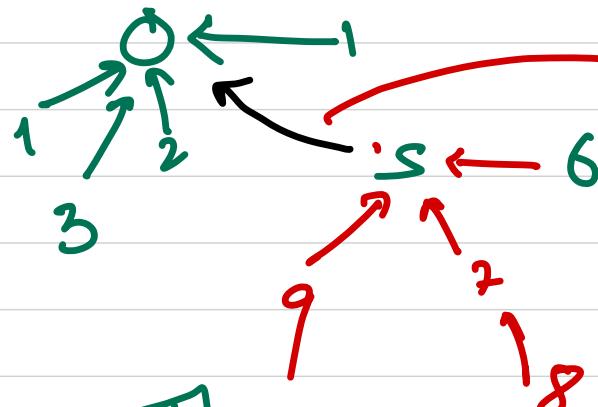
$\rightarrow \text{get} \rightarrow \underline{\underline{O(1)}}$

$\rightarrow \text{unions} \quad \underline{\underline{O(\log n)}}$  //

Rank → level

run by - an

sum size



what is the new  
link added?



an array of about  
sum size



-log 1

path compression

→ inverse Ackermann → constant

$O(\log^{*_n})$  → extremely slow  
growing func<sup>n</sup>

$\log^{*_n} \rightarrow$  no. of steps we need to take

$\log_2^n$  as the value  $n$  to make it  
smaller than one.

$$\log(2^{16}) \rightarrow \underline{\underline{16}}$$

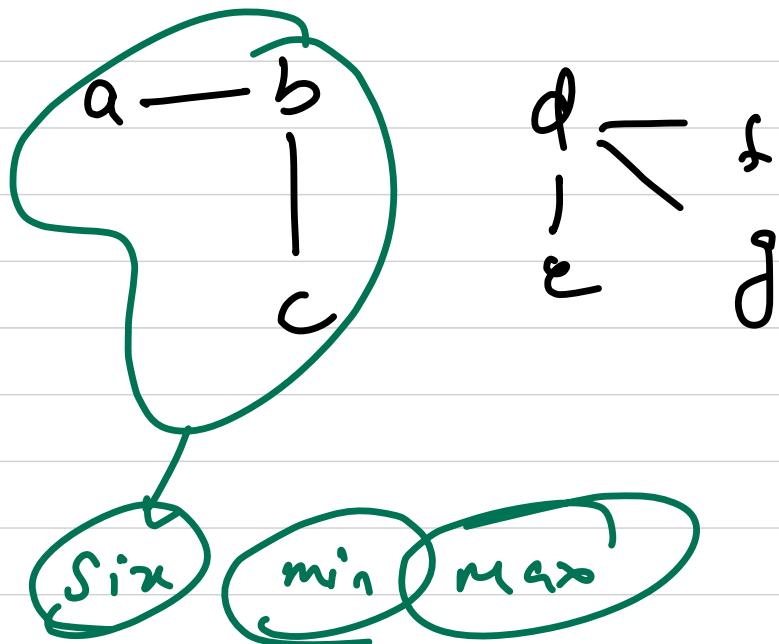
$$\log \underline{\underline{2^{16}}} \rightarrow$$

$$\log 2^{16} \rightarrow \log 2^4 \rightarrow \log 2^1 \rightarrow \log 2^0 = 0$$

1

# connected component

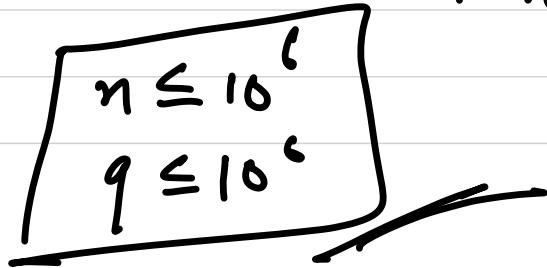
$$\rightarrow \frac{O(v + e)}{O(\log^* v)}$$

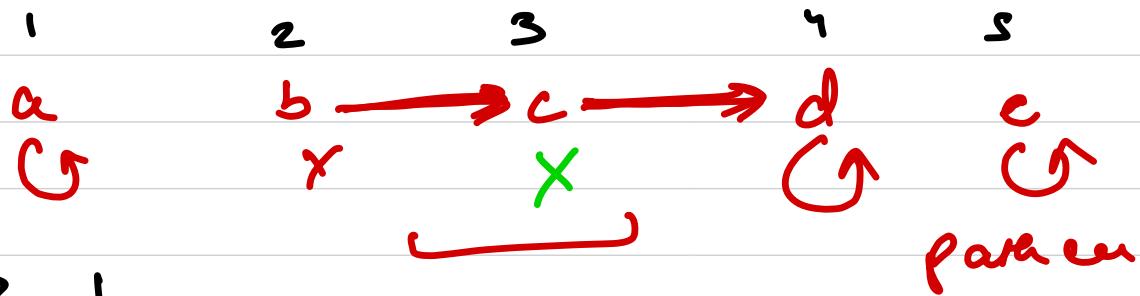


~~d: amnd  
d: is not~~

~~if~~  $\Rightarrow$   $n$  persons standing in a row at positions  $[1-n]$ . We can do 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 (not removed).





$? 1 \rightarrow 1$

$- 3$

$? 3 \rightarrow 4$

$- 2$

$? 1 \rightarrow 1$

$? 2 \rightarrow 4$

⋮

DSU

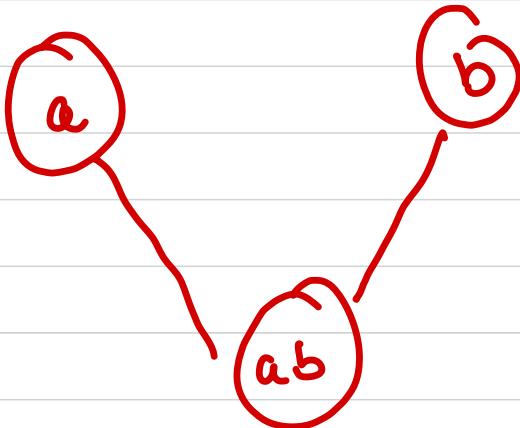
unite ( $x+1, x$ )

$? x$  → get(x)

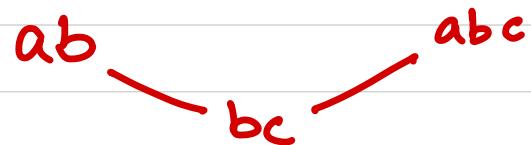
~~Q21~~

Secret Password - ~~Codeforces~~ → ?  
DSU?

DFS

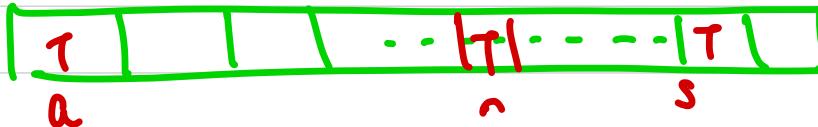


~~graph  
components~~

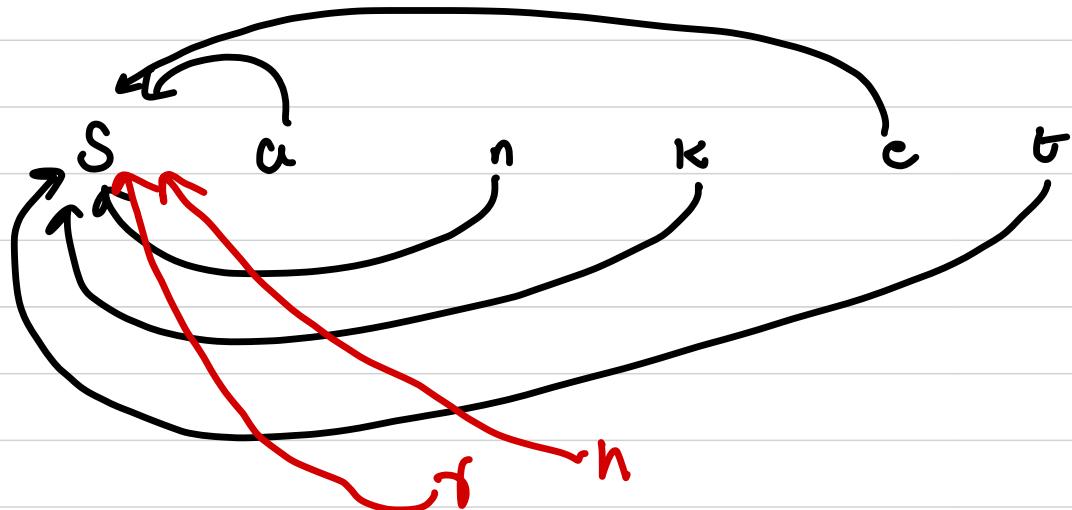


~~Q21~~

26



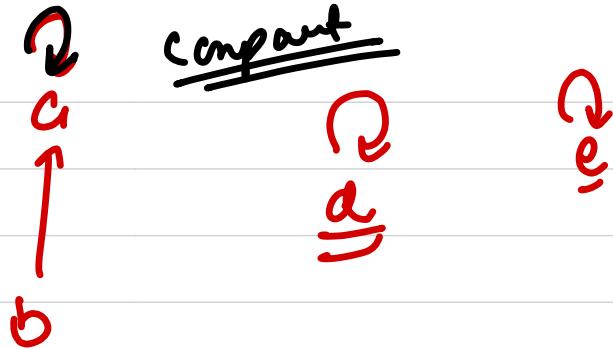
Sanket



as thak

27

a  
b  
 $a \rightarrow b$   
d



Total nodes  $\rightarrow \underline{\underline{26}}$

1 node  $\rightarrow \underline{\underline{1 char}}$

# of nodes   parent( $i$ ) =  $= i$     $\rightarrow$  # of compant.

G = Given an undirected graph. Some edges are given,

remove  $a, b \rightarrow$  remove edge from  $a \sqcup b$

get  $a, b \rightarrow$  tell if  $a \& b$  are in same component:

$$V \leq 3 \times 10^4$$

$$E \leq 10^9$$

$$q \leq 15 \times 10^4$$



After all removals there is no edge left

# Online query vs Offline query

$q \rightarrow l_1, r_1$



immediately

$q - l_1, r_1$

$l_2, r_2$

$l_3, r_3$

$\vdots$

$l_n, r_n$

stone

work

→ DSU → union - find

~~offlengay~~

$$\frac{c,c}{\cancel{a} \ b} \rightarrow T$$

- a b

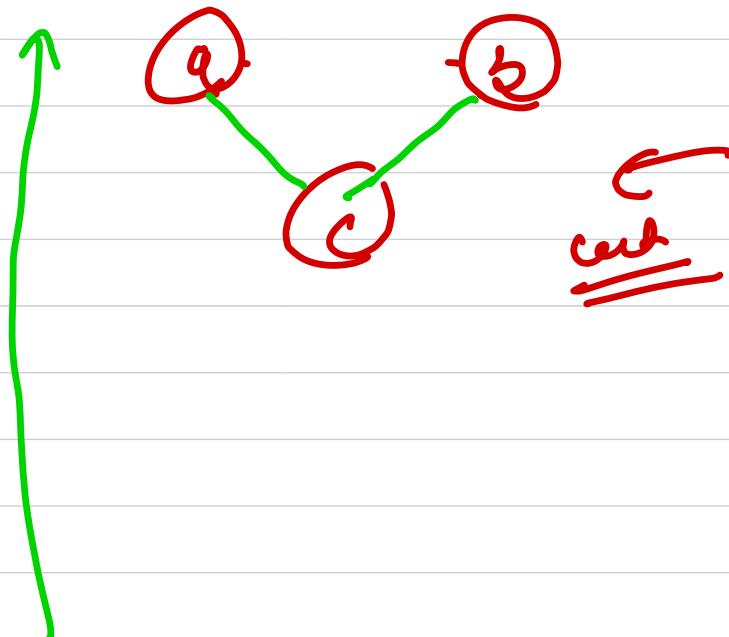
a, b → T

- a, c

b, a → f

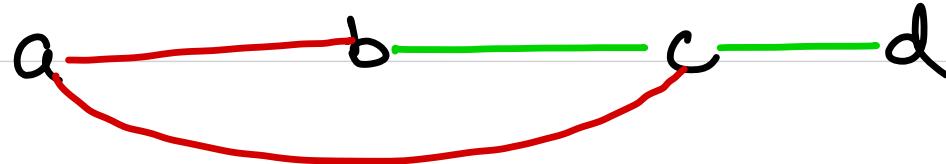
- b, c

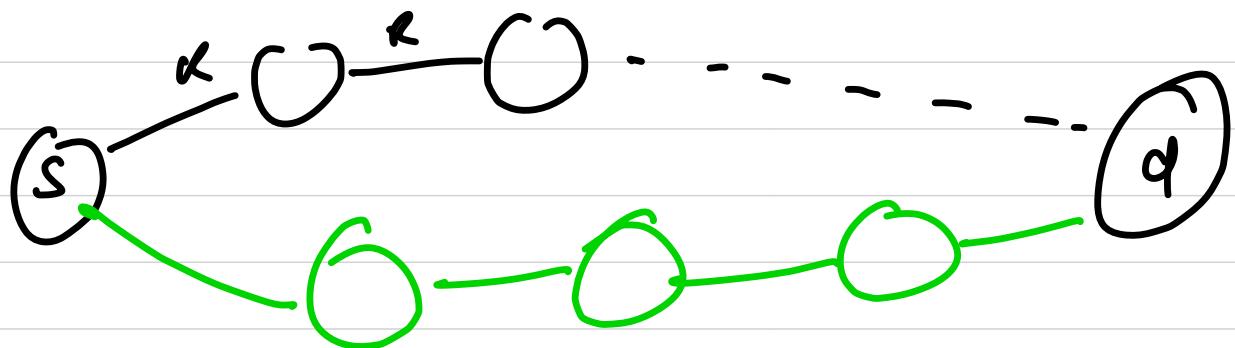
a b → f



P<sub>n</sub> Given a graph with  $v$  nodes &  $e$  edges,  
every edge has an associated color w it either  
**Red** or **Green**. Given a src & dest, find the  
shortest path such that your path starts  
from red edge & ends at green edge &  
we can switch from a red to green edge only

ans -





P: We have a new lang that uses english  
chars. The order of char is unknown. Given a  
list of strings of new lang, sorted lexicographically  
by rules of new lang. Return a string of unique  
letters in the new lang sorted in lexicographical  
order.

[ "wøt", "wøf" ]

↳ wøtf ↳

$a b c$

$a < b < c$

$a < d < e$

$b < d$ .

$b < c$

$d < e$

$d < e$

$a d e$

inequalities

Dependencies

~~Q~~ Alien-Dictionary → Leetcode

english → a ← starts with

( wst, wtf, cr, ett, rftt )  
↓      ↓      ↓      ↓      ↓  
w,    w,    c,    e,    t

f, t

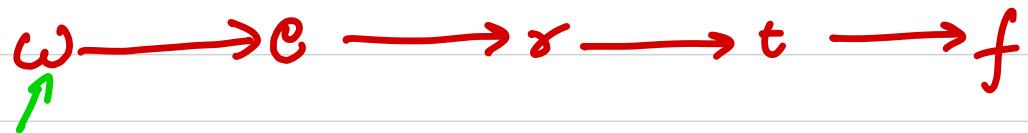
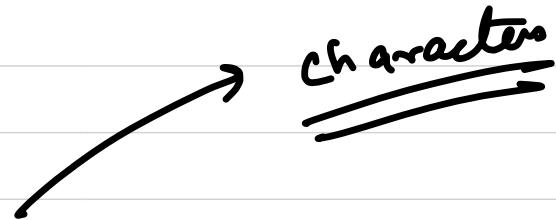
w, e, t ⇒ w < e < t

wst, wtf ⇒ t < f

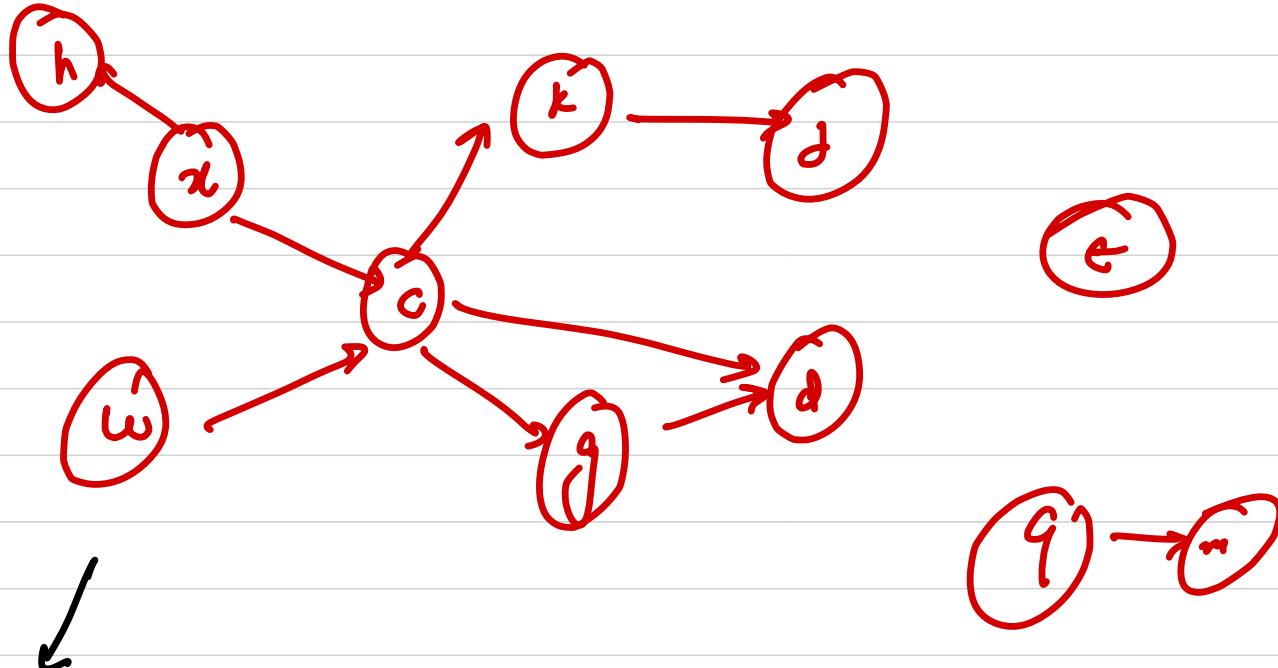
r < t

→ dependency graph

w < e < s



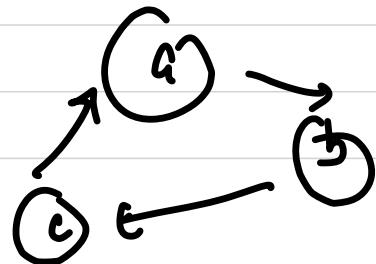
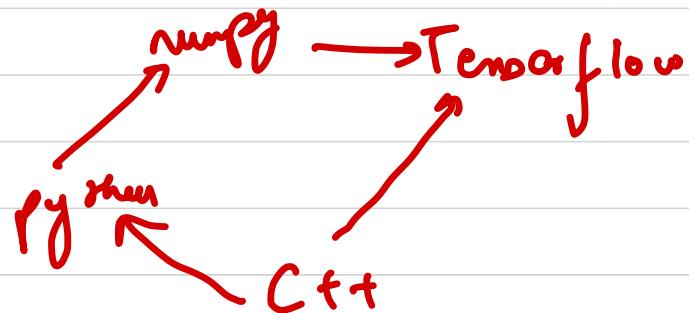
Wertf



Complex dependency graph → a dep of clubs

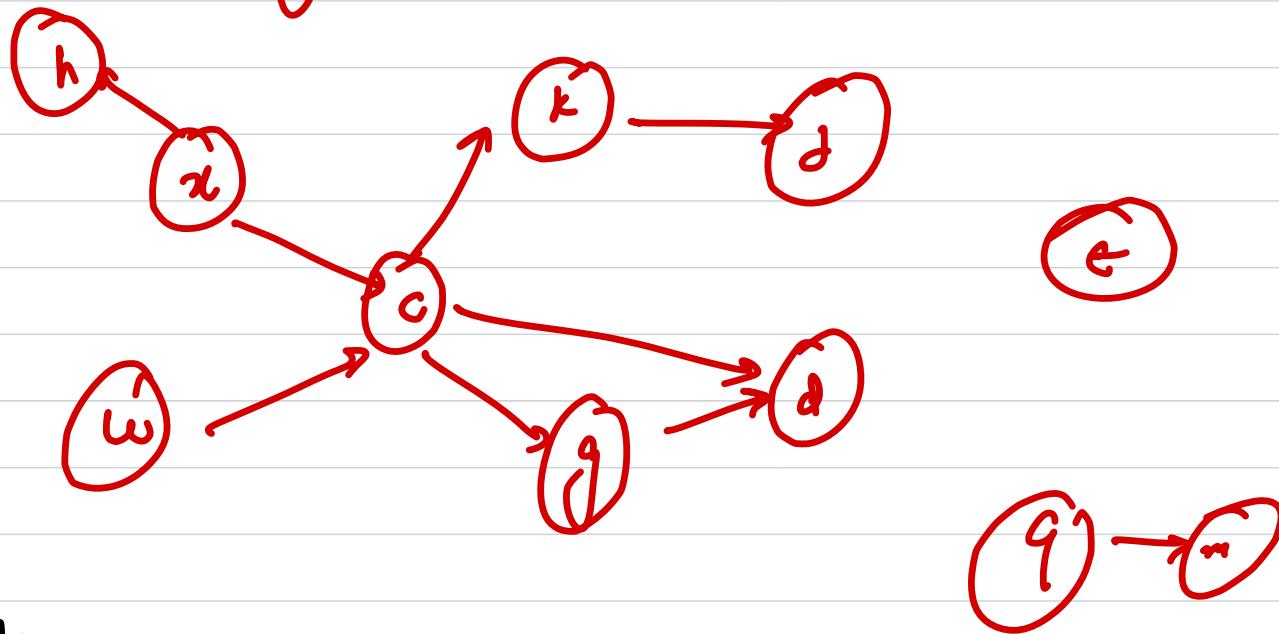
# Topological Sorting

C++, Python, numpy, TensorFlow

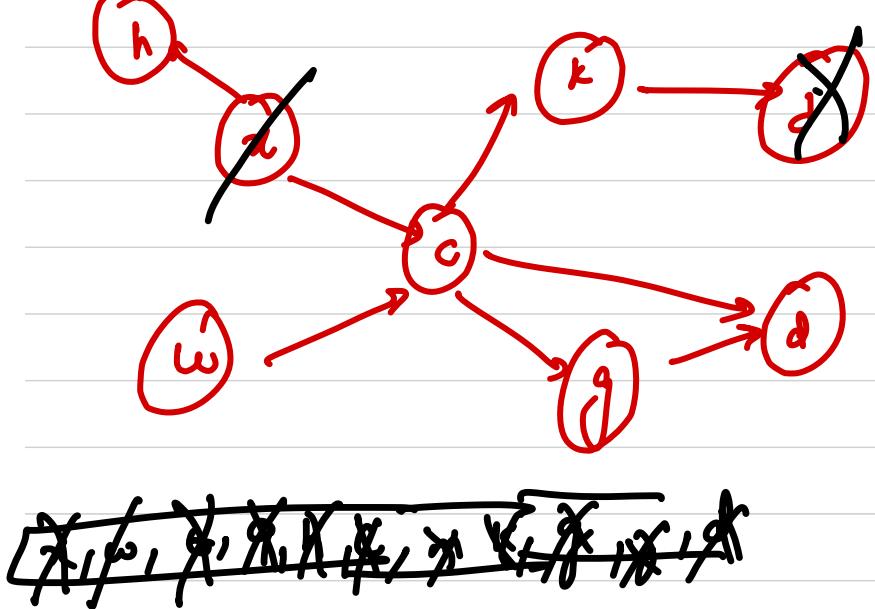


# # Kahn's Algorithm

(modified bfs)



Resolve those first who have no dependency } Direct



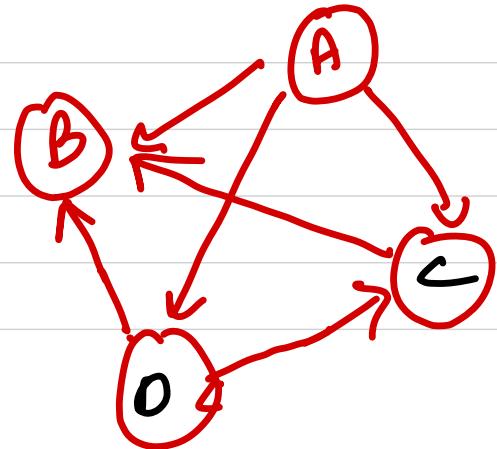
$$g(v) \cdot p_6(v)$$

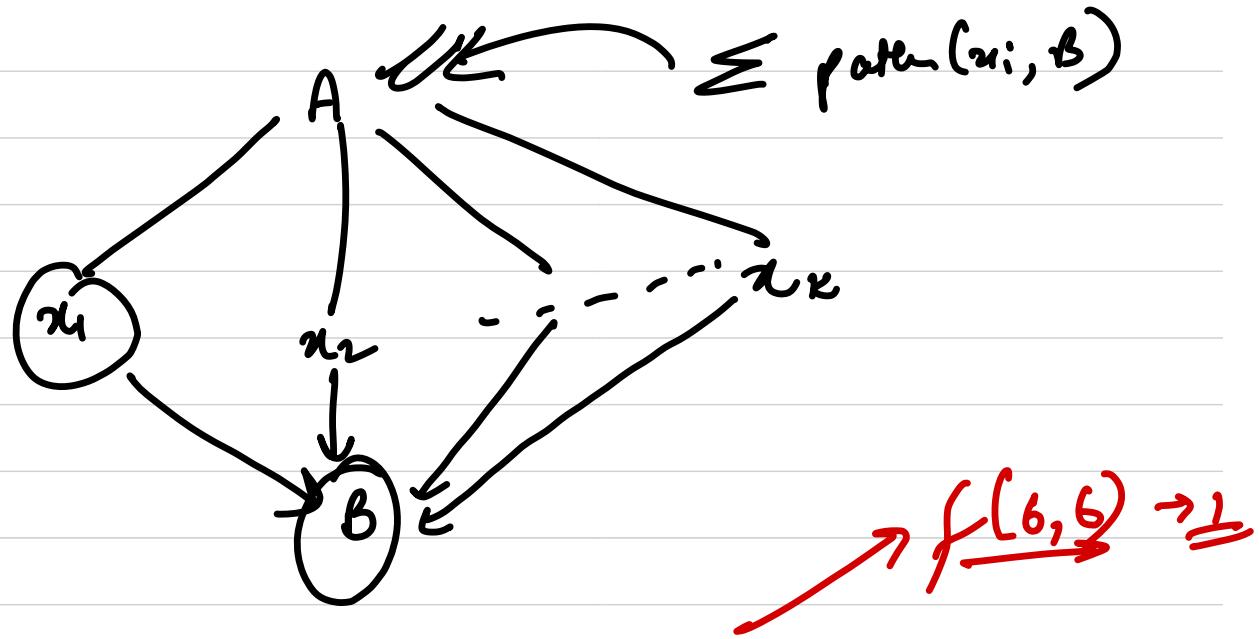
VTT

$\pi, w, t, q, h, c, m, k, g, j$

undergo → u, n, d, g, e, g, o, u, g, o, u, g, o, u, g, o, u, g, o

~~Q~~ Given a directed acyclic graph with  $\sqrt{v}$  vertices & 'c' edges . find the no. of paths that exists between any given node A to another node B.



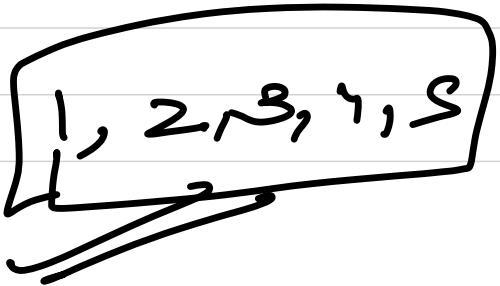
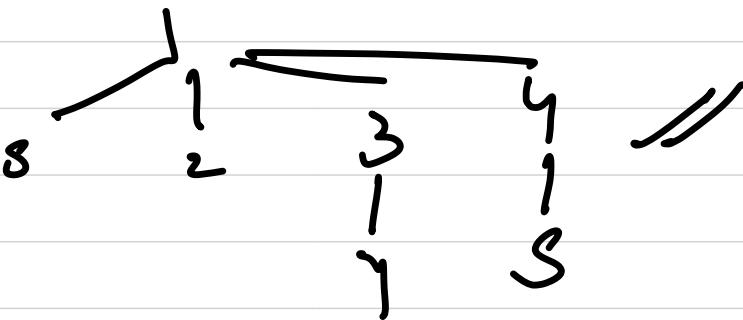
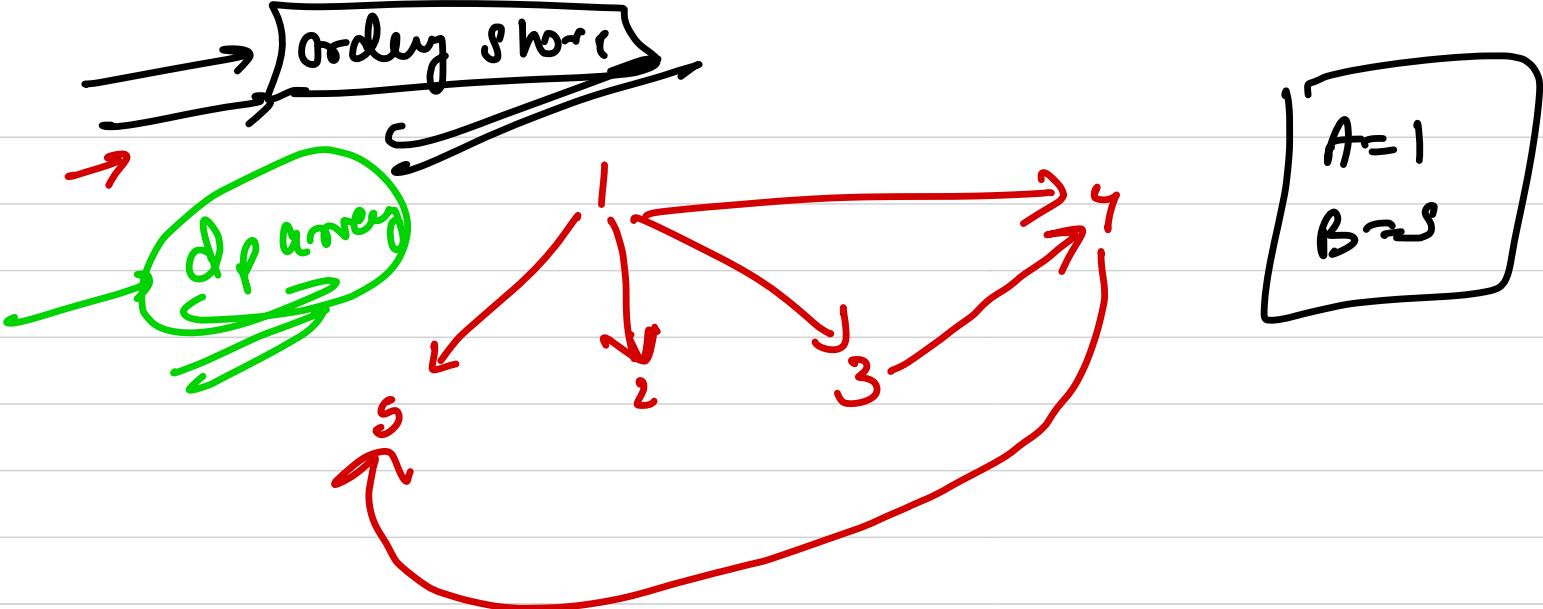


$$f(v, v) = \sum f(x_i, v)$$

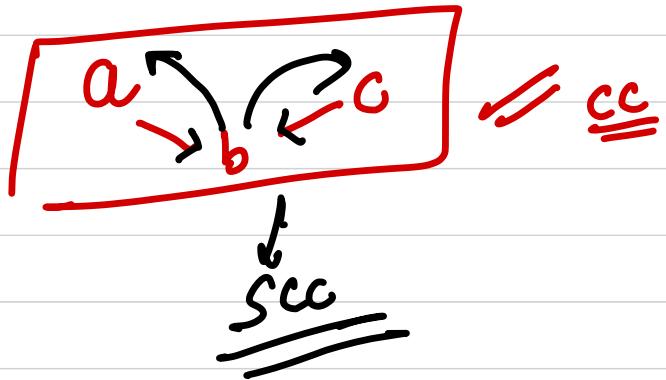
# of path from  $v$  to  $v$

$x \in \underline{\text{neighbor of } v}$

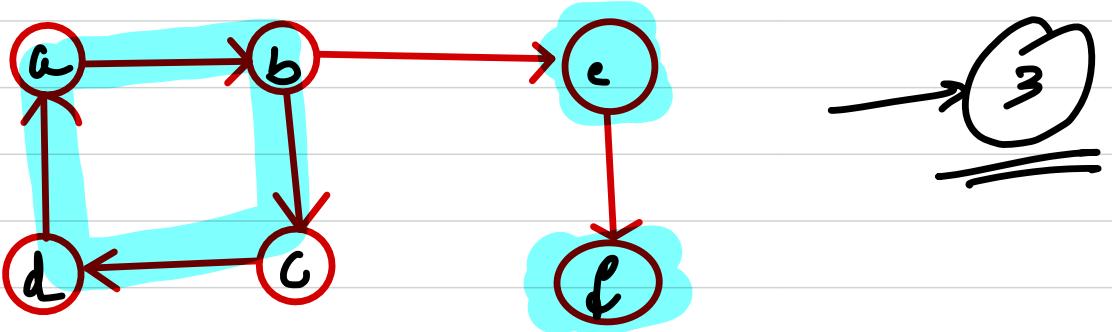
$f(a, b) \geq 1$

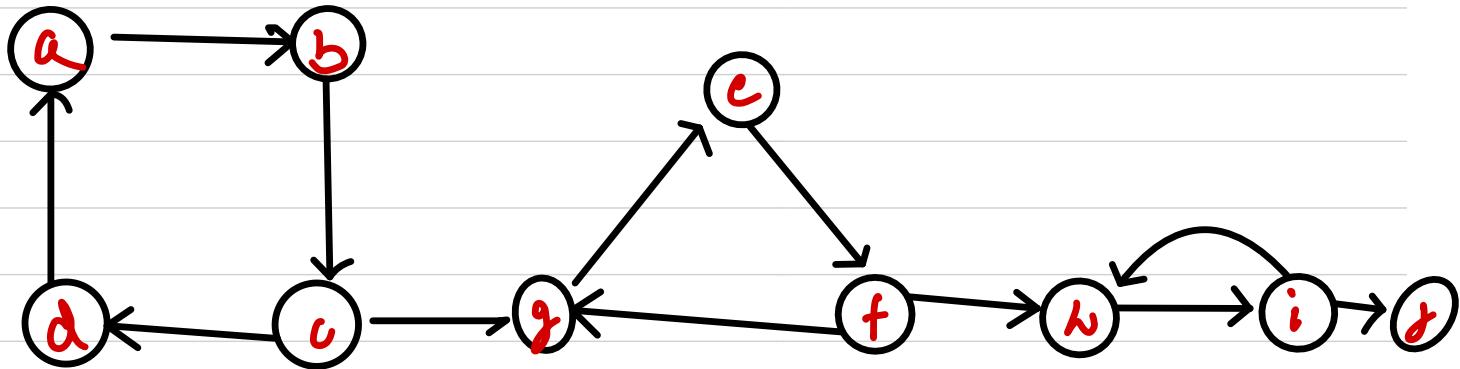


# # Strongly Connected Component.

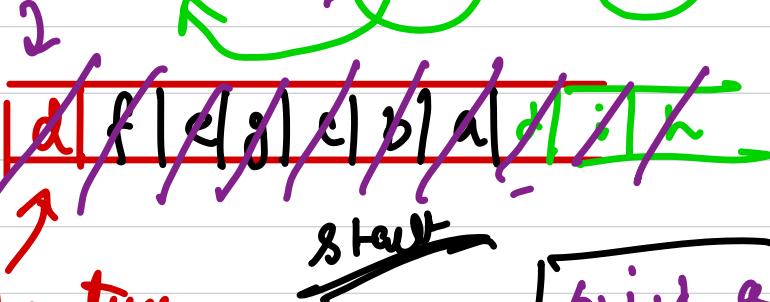
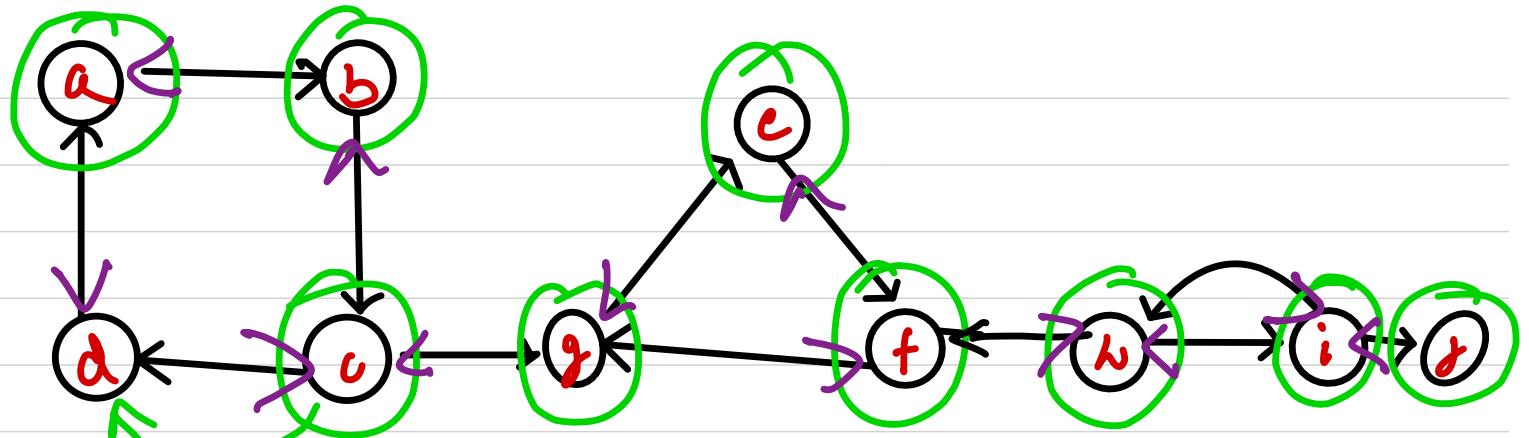


How many strongly connected Components are =





Kosaraju Algorithm → Used to find SCC



perform DFL, & strong  
nodes when there don't  
have unwanted neighbor

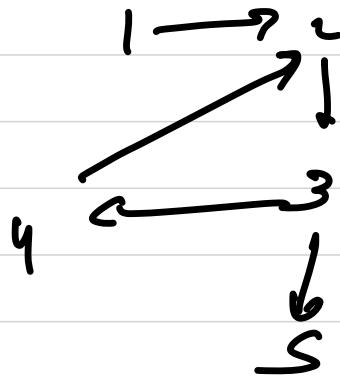
vis =

a, b, c, d
g, e, f, h
i, j

vis2

i, j, a, b, g, h
e, f

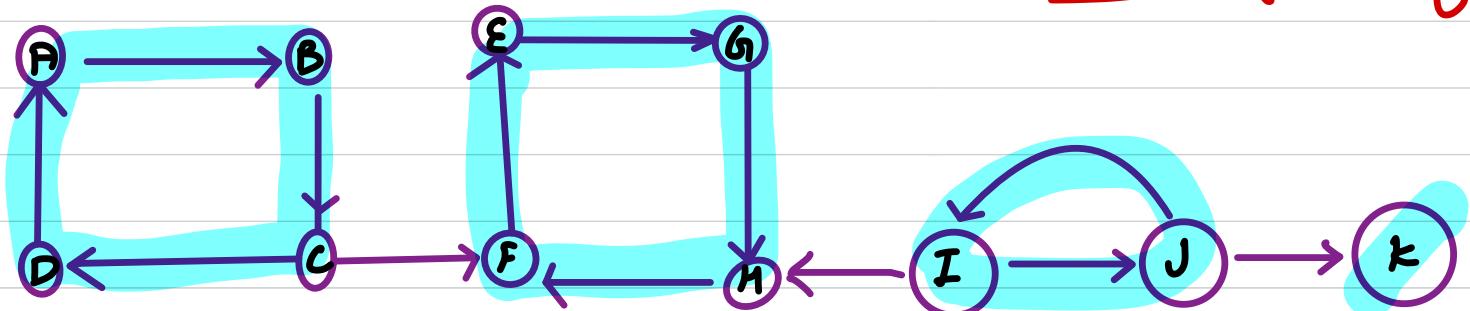
Transpose the graphs



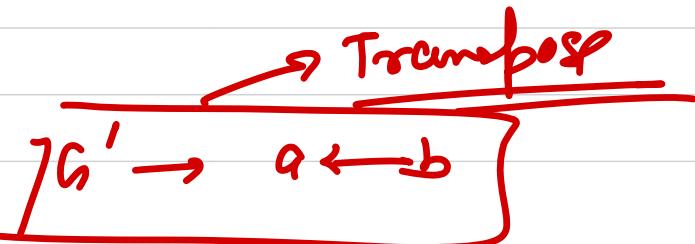
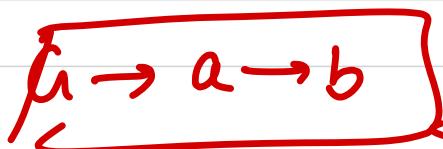
1, 2, 3, 4

Strongly Connected Component

~~Kosaraju's Algo~~



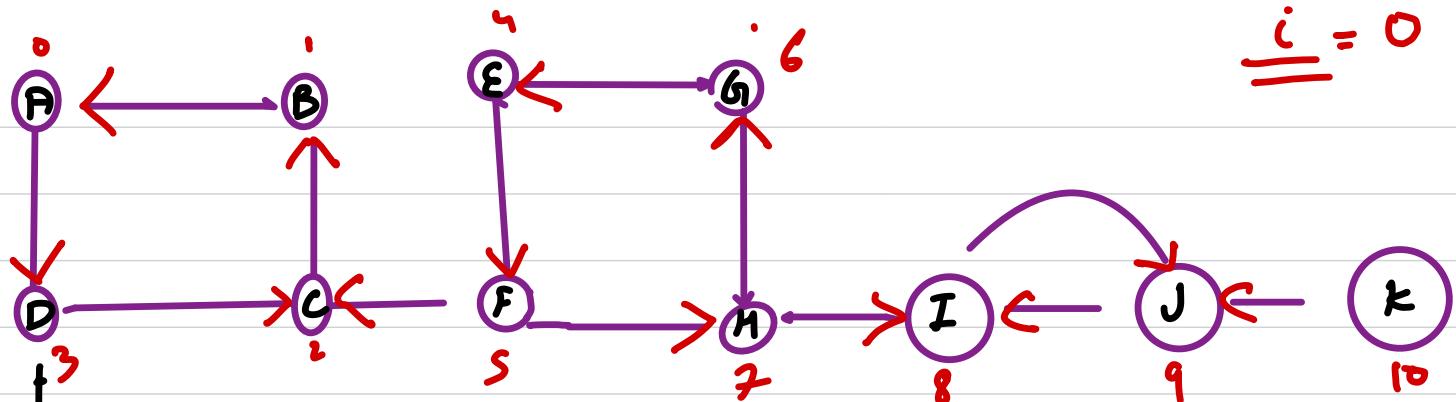
$$G \rightarrow G'$$



Two phases

- 1) Before Transpose  $\rightarrow$  We start a dfs, to find all vertices reachable from a node ( $v$ )
- 2) After transpose  $\rightarrow$  We check the reverse i.e. find the subset of nodes that can node ( $v$ )





A B C D  
 f H G E

I J

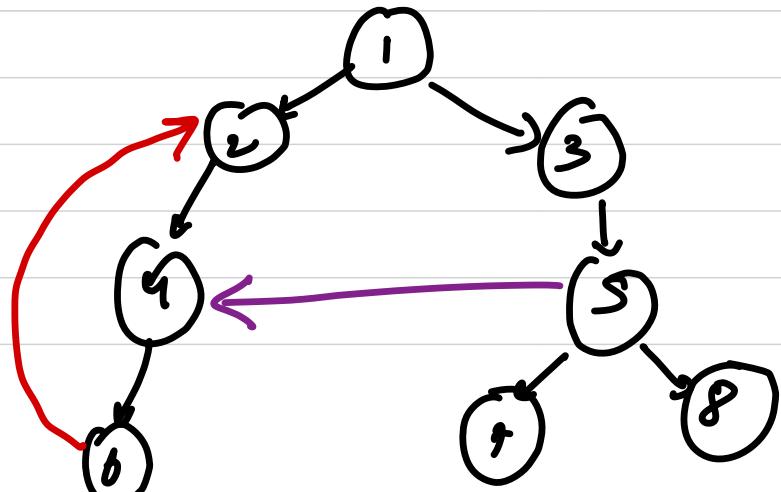
K

I, J, K, a, d  
 c, b, f, m, g, e

Visited

You've a directed graph G<sub>1</sub>. If we do dfs on G<sub>1</sub>, we will get forest of trees.

We get a set of trees due to dfs, order can be diff.



Back edge  $\rightarrow$  it's an edge from  $(v, u)$  such that  
 $v$  is ancestor of node  $u$  but not part of  
dfs tree.

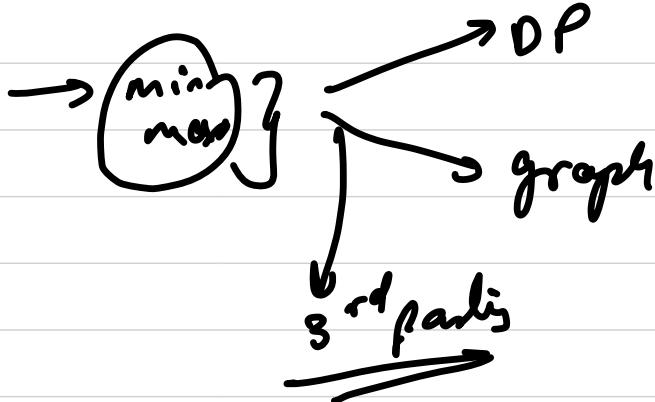
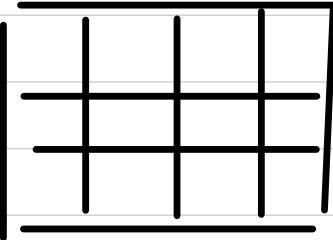
Cross Edge  $\rightarrow$  It connects two node such that  
they do not have any ancestor & a  
descendant relation between them

  
A cross edge is seen from a lower ordered node to a higher order node.

Suppose there is a cross edge from lower ordered node  $v_1$  to a higher ordered node  $v_2$ . Then

during the DFS on  $v_1, v_2$  must have been visited.

& we would have called DFS for  $v_2$  in this order. Then in that case  $v_2$ 's order is less.



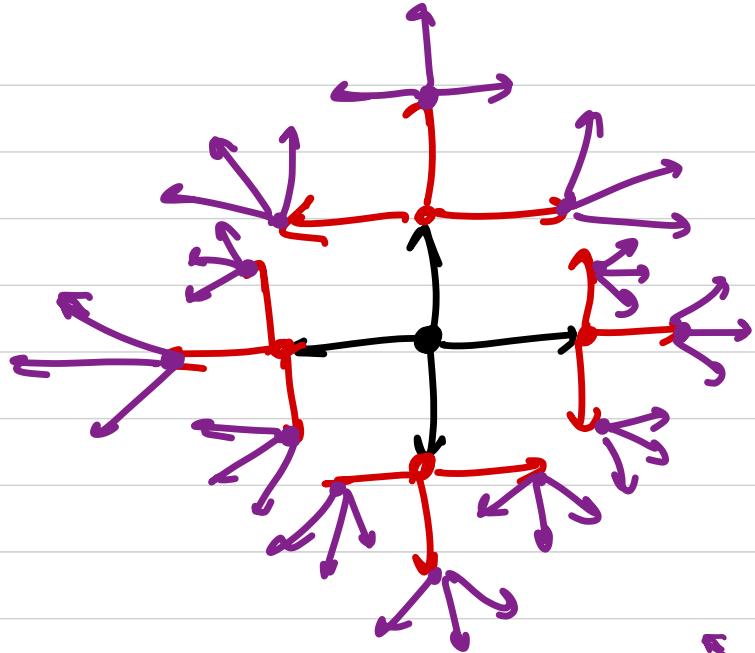
Can this be  
solved by  
graph

Multi Source BFS

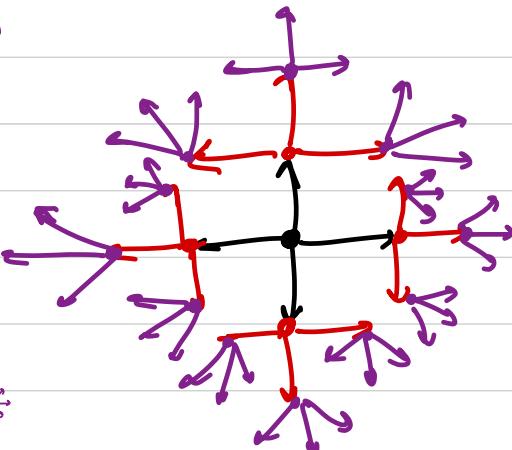
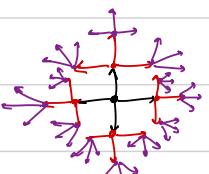
level

↓

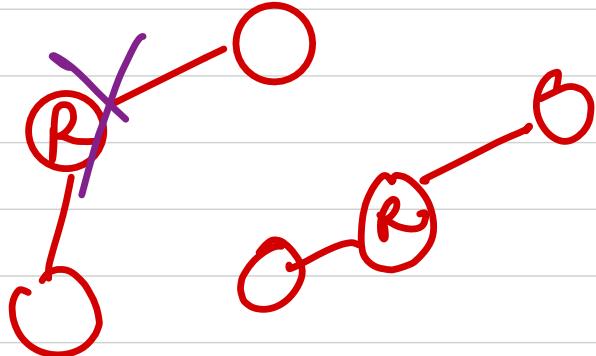
off



	O	O	
O	X	O	
O			R
O			O



minlen = the sum  
instance when  
last crap is  
gets rotten



push all the sources in que

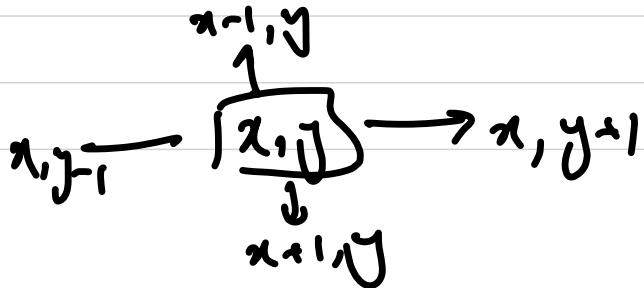
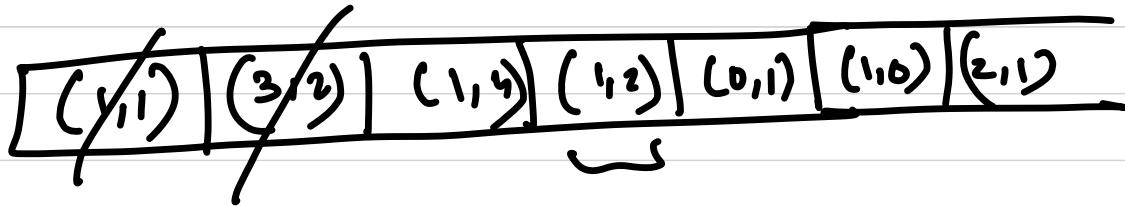
	0	1	2	3	4
0	0	b			
1	0	R	0		R
2	0			0	0
3	0	R	0	0	0
4					0

stapel von den  
deutet

node  $\rightarrow \langle x, y \rangle$

Set  $\langle$   $\rangle$

$\langle x, y, l \rangle$



~~Q~~ Given a grid of chars,  $\{L, R, U, D\}$ . The char at  $(i, j)$  cell denotes the direction in which you can move from  $(i, j)$ . You start from  $(0, 0)$ , check whether it is possible to reach  $(n-1, m-1)$  or not ?.

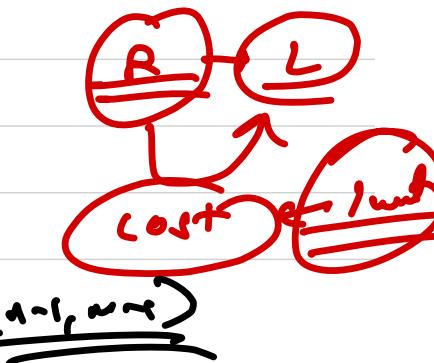
R	R	D
U	L	D
R	R	R



Swap

min cost

$(0, 0) \rightarrow (n-1, m-1)$



$\swarrow \rightarrow$  Banteforum under pressure (Diffs)

Worst Case  $\rightarrow TC \rightarrow O(nm)$

$SC \rightarrow O(nm)$

Fourwise-hard approach

	R	R	D
	D	L	L
	R	R	U

Space =  $O(1)$

2 2

worst case  $\rightarrow$   
layer of salt  $\rightarrow nm$

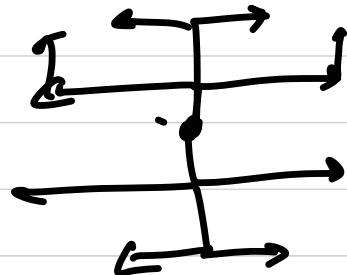
step counter

$\rightarrow$  C  $\leftarrow$  BOS  $\rightarrow$  LL  $\rightarrow$  cycle detector

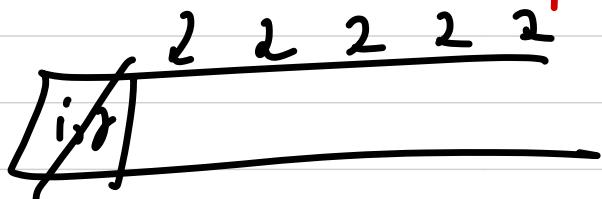
# Knights on a chess board

code 2.1

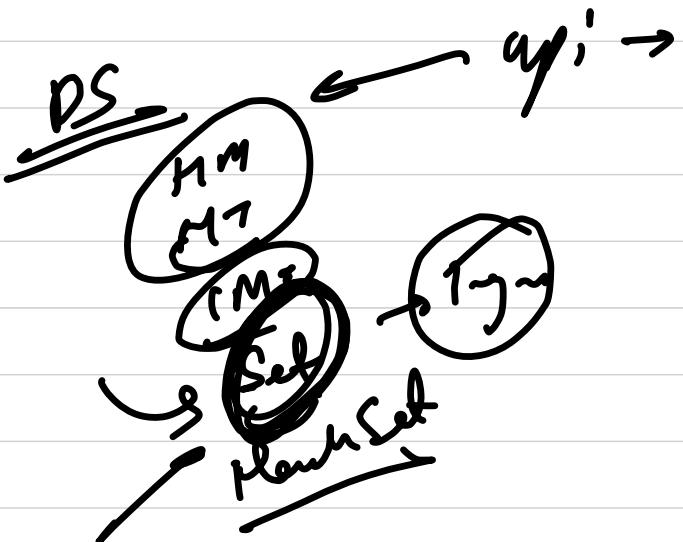
BFS



$(i, j)$



linked list



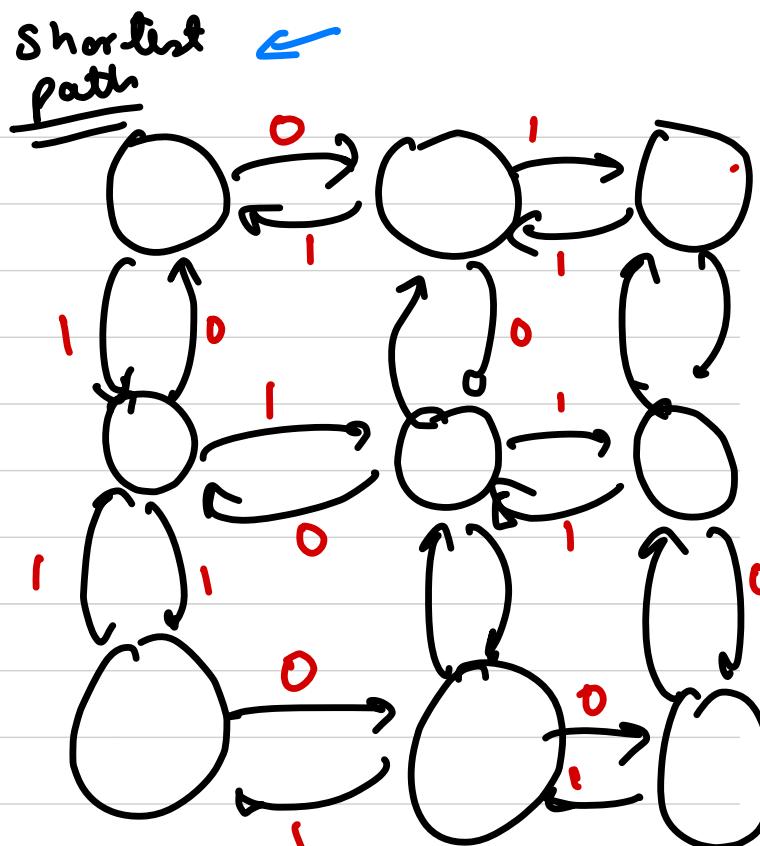
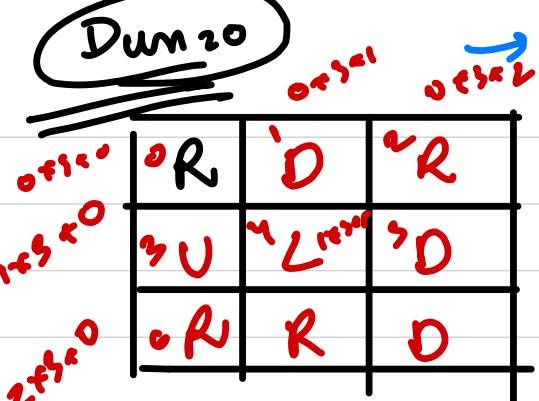
String methods

$p_0 = b_0$

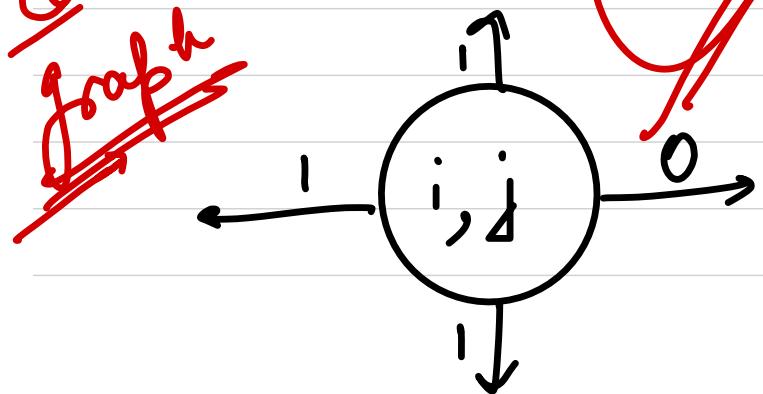
my Sq1

The diagram shows a box containing the assignment statement  $p_0 = b_0$ . Below the box, there is an arrow pointing to the word "my" followed by "Sq1", which is underlined.

~~Max~~  
~~BFS~~



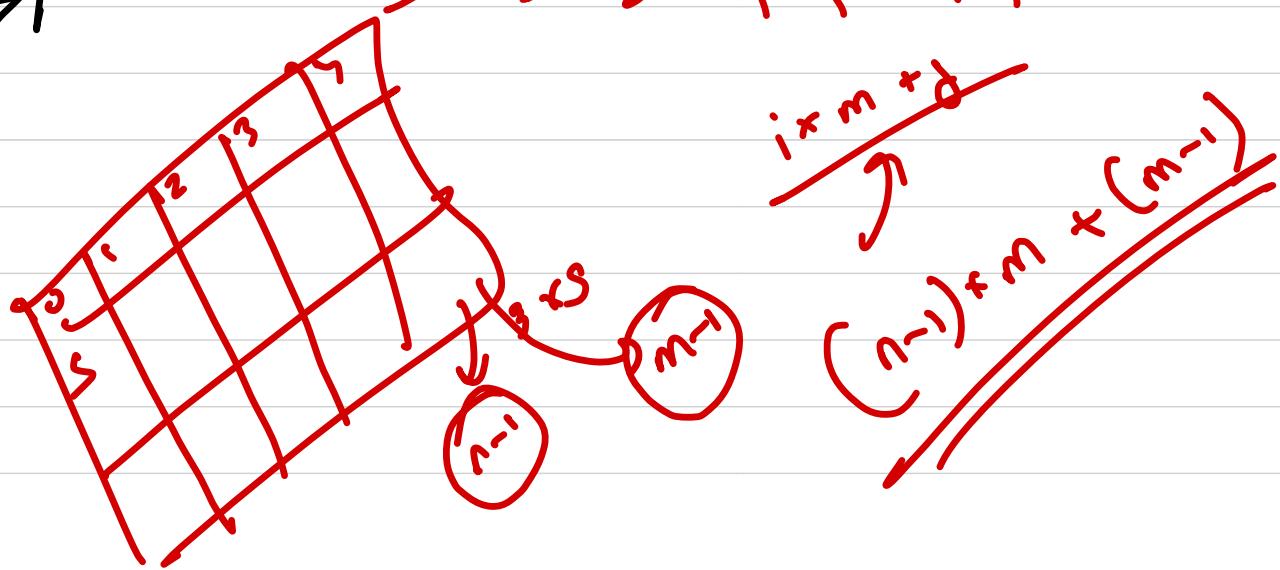
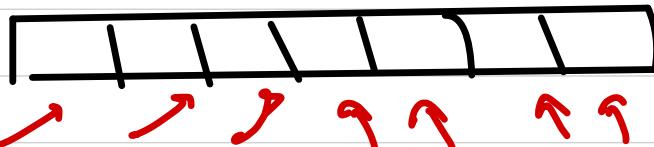
Construct a graph



0-1 Bfs

gscudy

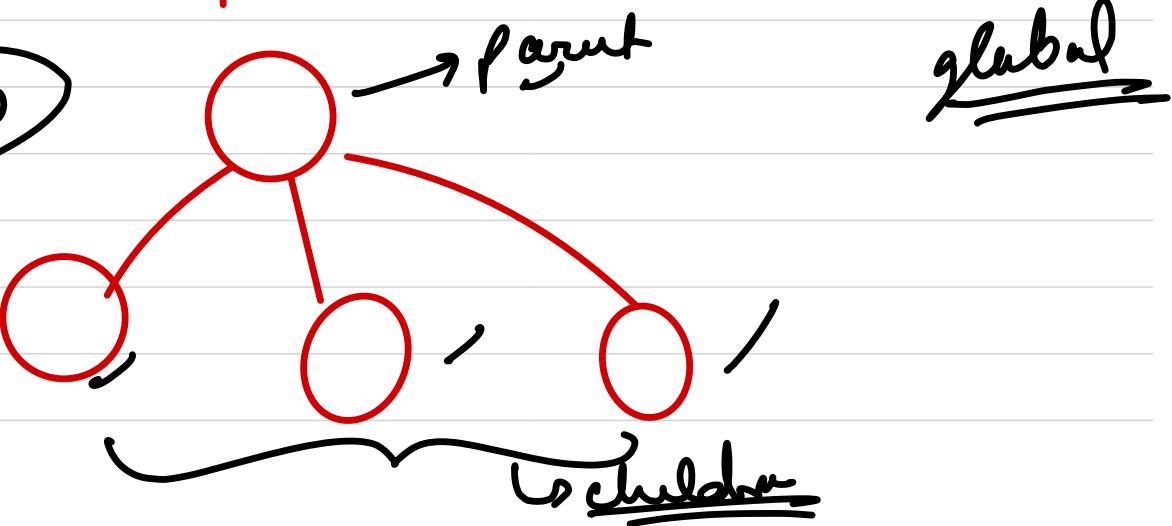
deque



## → D P on trees

Primary axis  
Secondary axis

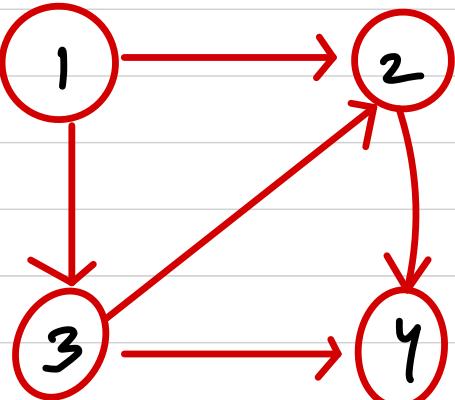
- I → Pattern 1
- " → Pattern 2



longest path  $\rightarrow$  at code

$$f(n) \rightarrow 1 + \max(f(n-1))$$

~~ans~~  
 $\max_{i \in \{n\text{ nodes}\}} (f(i))$

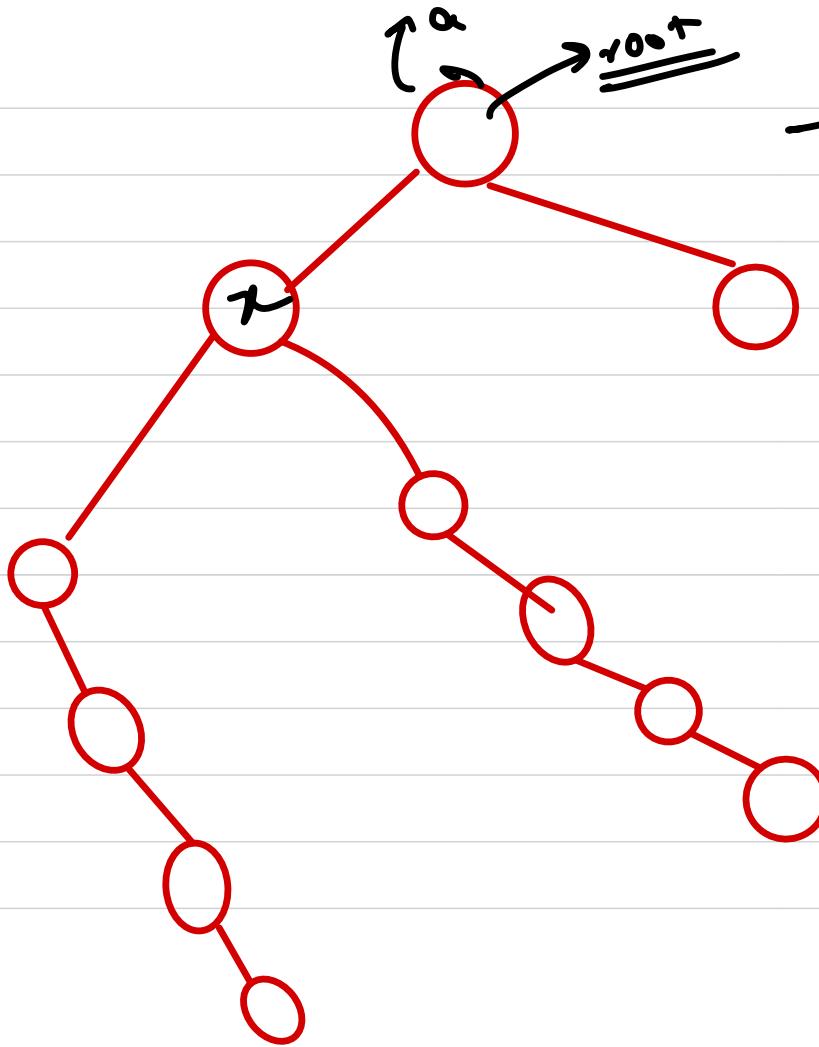


Tree

Ans  $\rightarrow$  3

11

Binary Tree  $\rightarrow$  Diameter



$\alpha$   
100<sup>T</sup>

Binary tree

$\rightarrow \delta \text{ and } \epsilon \rightarrow l^T$   
 $\rightarrow \zeta, \eta, \theta \rightarrow r^T$   
 $\rightarrow \alpha, \beta \rightarrow \text{root}$

$$f(v) = g(v \rightarrow \text{left}) + g(v \rightarrow \text{right}) + 2$$

diameter  
of tree passing  
through v

$g(x) \rightarrow$  height of  $x$

$O(n^2)$

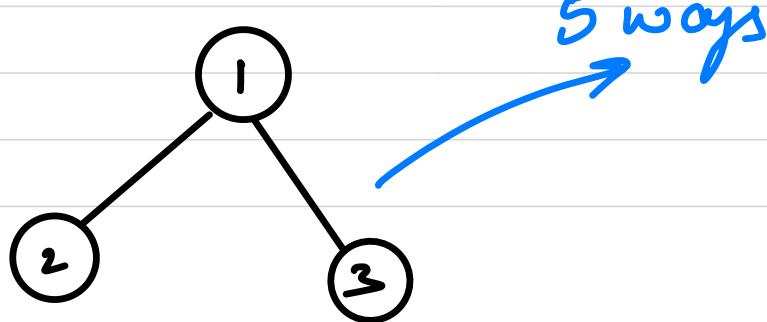
ans  $\rightarrow \max \underbrace{(f(v), f(v \rightarrow \text{left}), f(v \rightarrow \text{right}))}$

$$f(v) = \max(f(v \rightarrow \text{left}), f(v \rightarrow \text{right}) + 1)$$

~~height, darts~~  $\max(\text{left} \rightarrow d, \text{right} \rightarrow d, \text{left} + h + \text{right} + 2)$

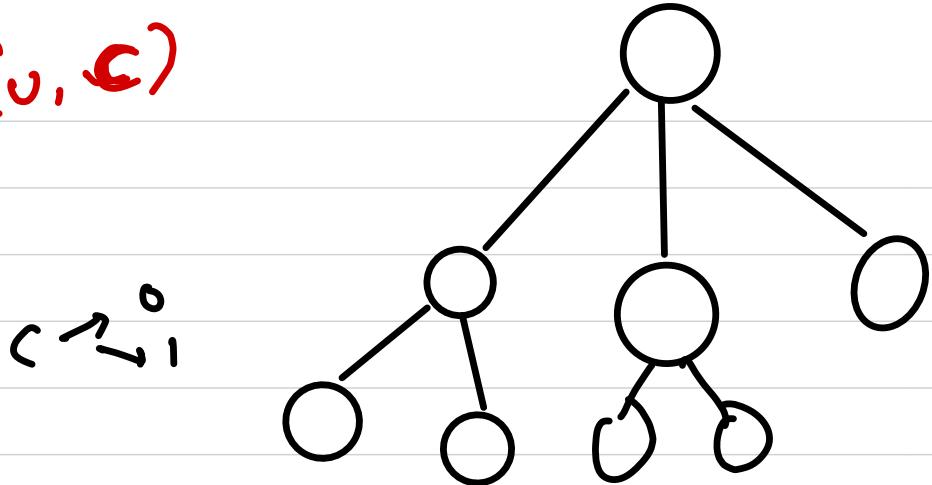
~~O<sup>n</sup>~~ You own a tree with  $N$  vertices. You need to do colouring of the vertices into cells blue or red. You can't color two immediate neighbors or adjacent vertices with blue. Find no. of ways to paint tree.

$$n \leq 10^5$$



$f(v, c)$

White  $\rightarrow$  Red  
Black  $\rightarrow$  Black



$f(v, 0) \rightarrow$  # of ways to color subtree rooted at  
 $v \rightarrow v$  can be B/R

$f(v, 1) \rightarrow$  # of ways to color subtree rooted at  
 $v \rightarrow v$  can be R

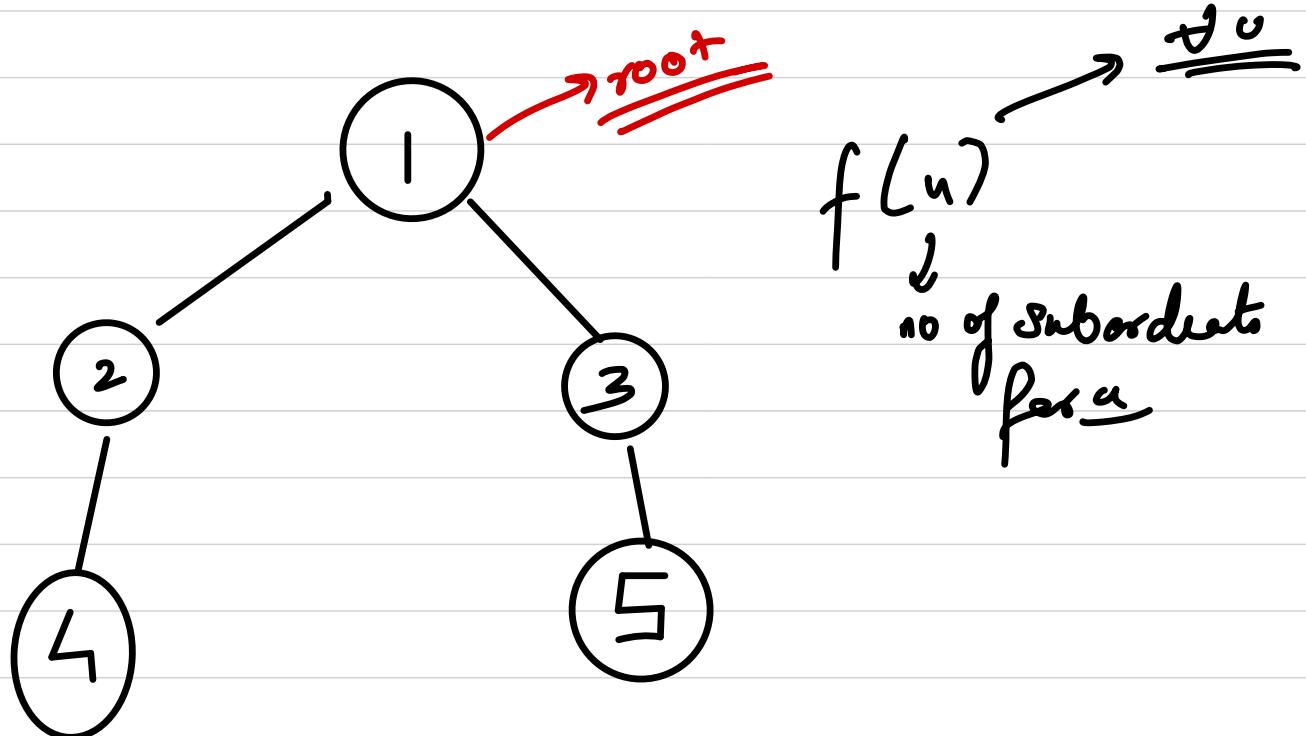
$$f(v, 0) = \pi f(c, 0) + \pi f(c, 1)$$

$$f(v, 1) = \pi f(c, 0)$$

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



\* Subordinates  $\rightarrow$  CSES



$$\underline{f(u)} = \sum (1 + \underline{f(c)})$$

df

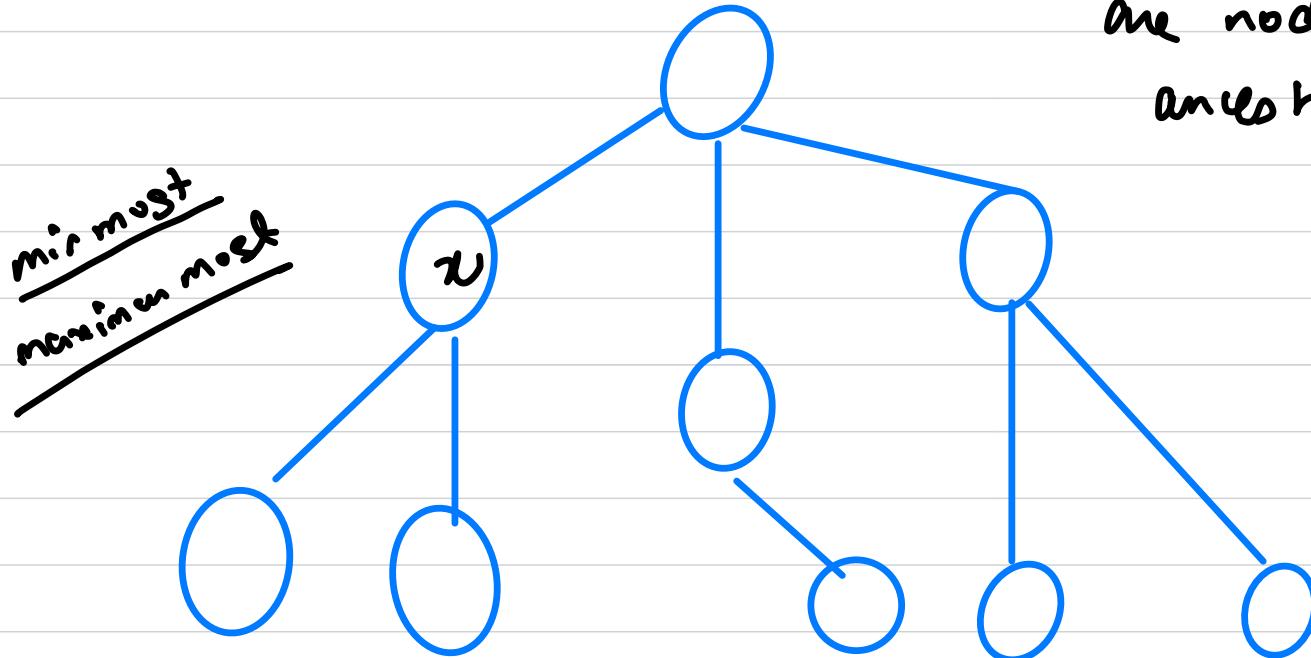


c ∈ children

f<sub>G</sub>

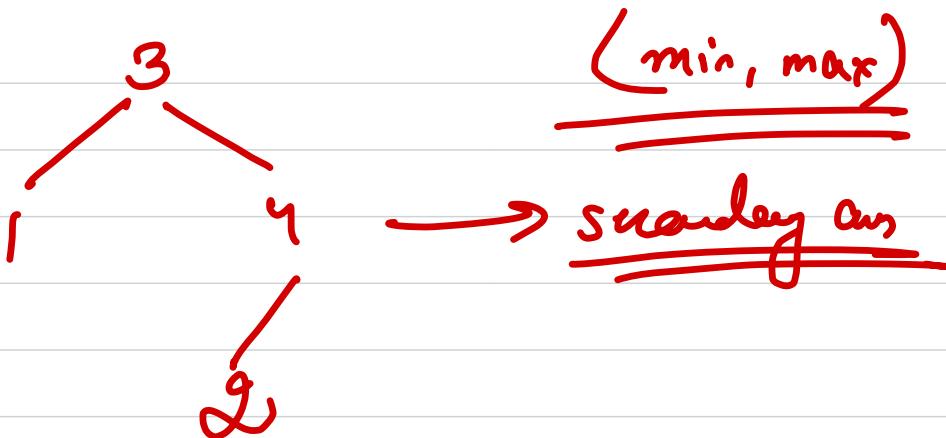
# farm tree → Codechef

mon diff such that  
one node is the  
ancestor of one



ans = -∞

(P, P<sub>c</sub>)



(min, max)

strategy ans

f(u) ⇒ (min, max) for the tree rooted  
at u.

$$\text{ans} = \max(\text{ans}, |\text{min}-u|, |\text{max}-u|)$$

$$f(u) = \{ \min(f(c)), \max(f(c)) \}$$



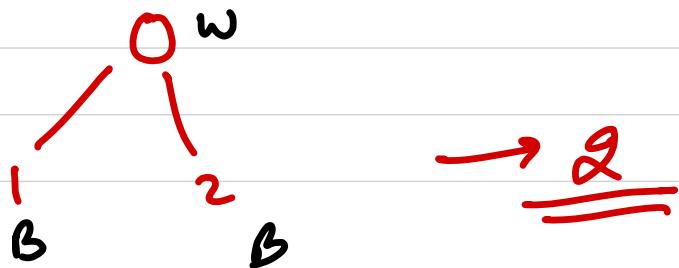
# apple man and tree → codforces

Divide this tree into arbitrary  $K$  components

such that each component has exactly one  
black node.

$$1 \quad 0 \leq K \leq n$$

Tree  $\rightarrow$   $n-1$

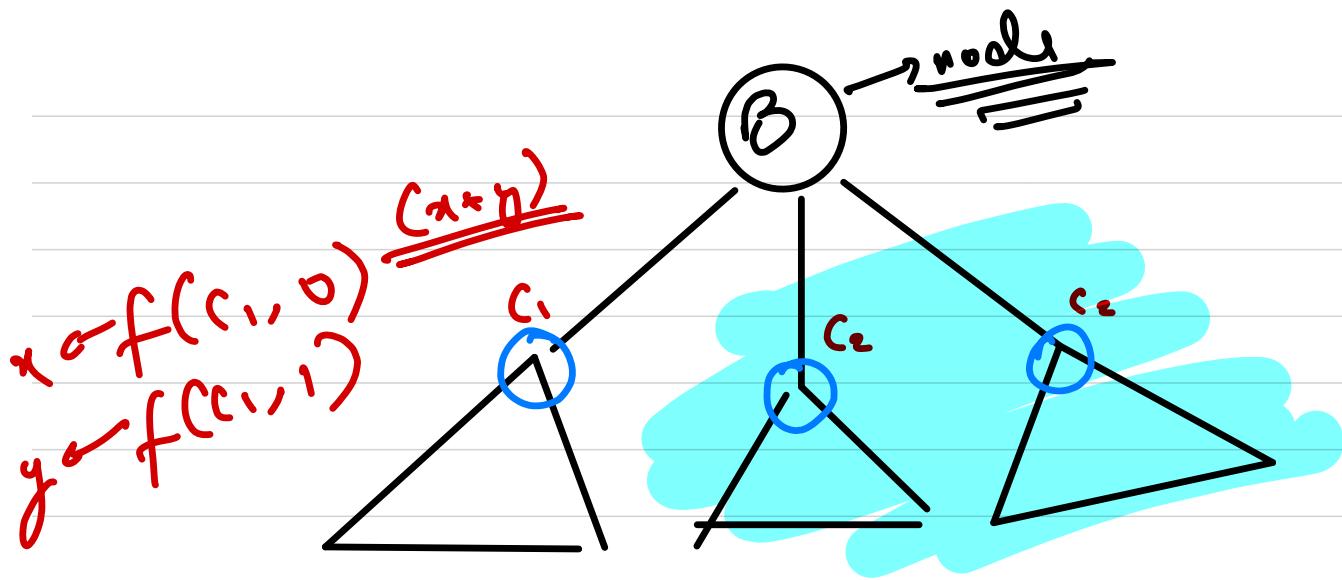


Every node will have different choices Based  
on colors of node in the Subtree

$f(\text{node}, c)$   $\xrightarrow{\text{count}}$

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

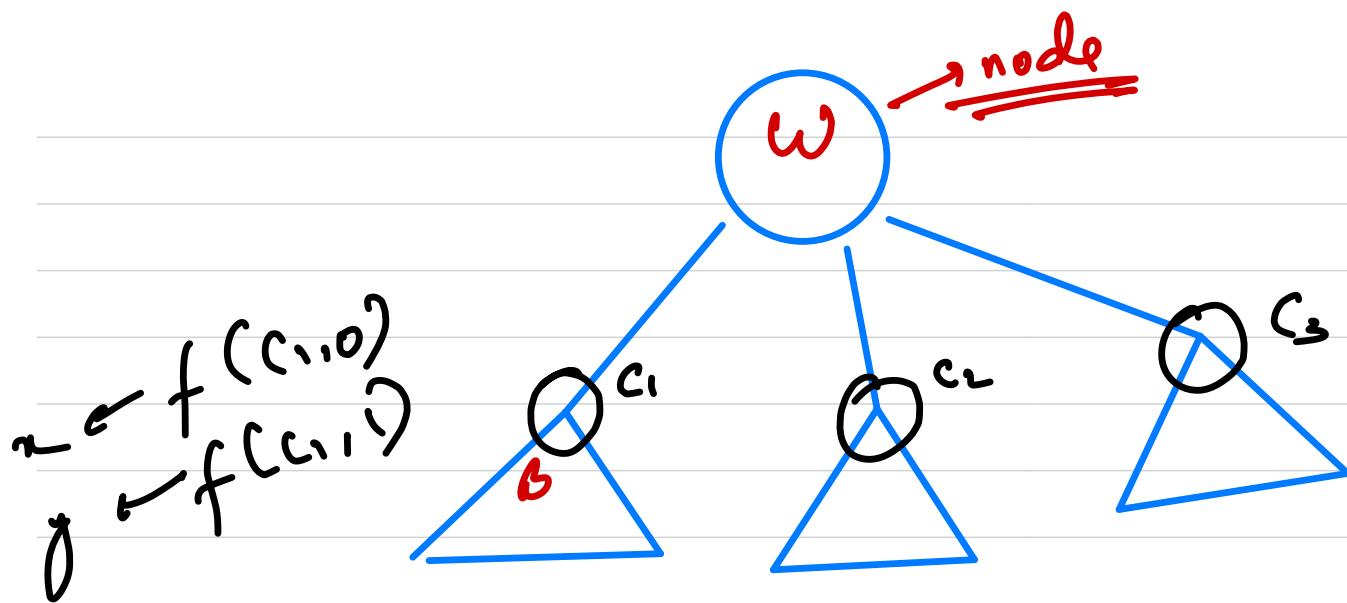
$f(\text{node}, 1) \rightarrow$  # of ways for subtree rooted at node  
 $\xrightarrow{\text{has 1 black vertex}}$



$$f(\text{node}, 1) = \bar{\pi} \left( f(c_i, 0) + f(c_i, 1) \right)$$

$c_i \in \text{child}$

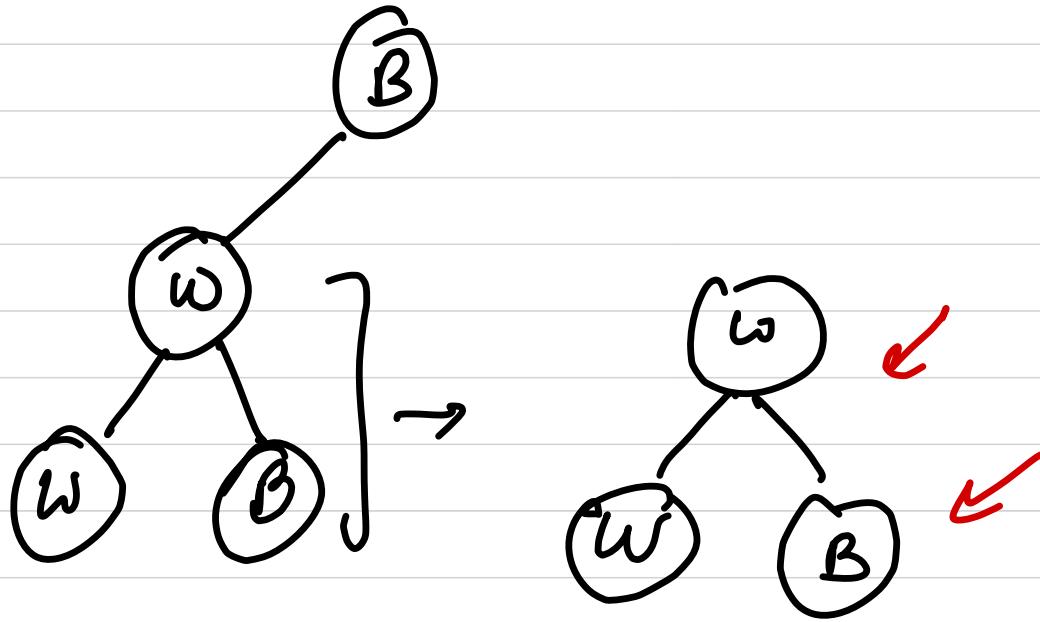
$$f(\text{node}, 0) = \underline{\underline{0}}$$



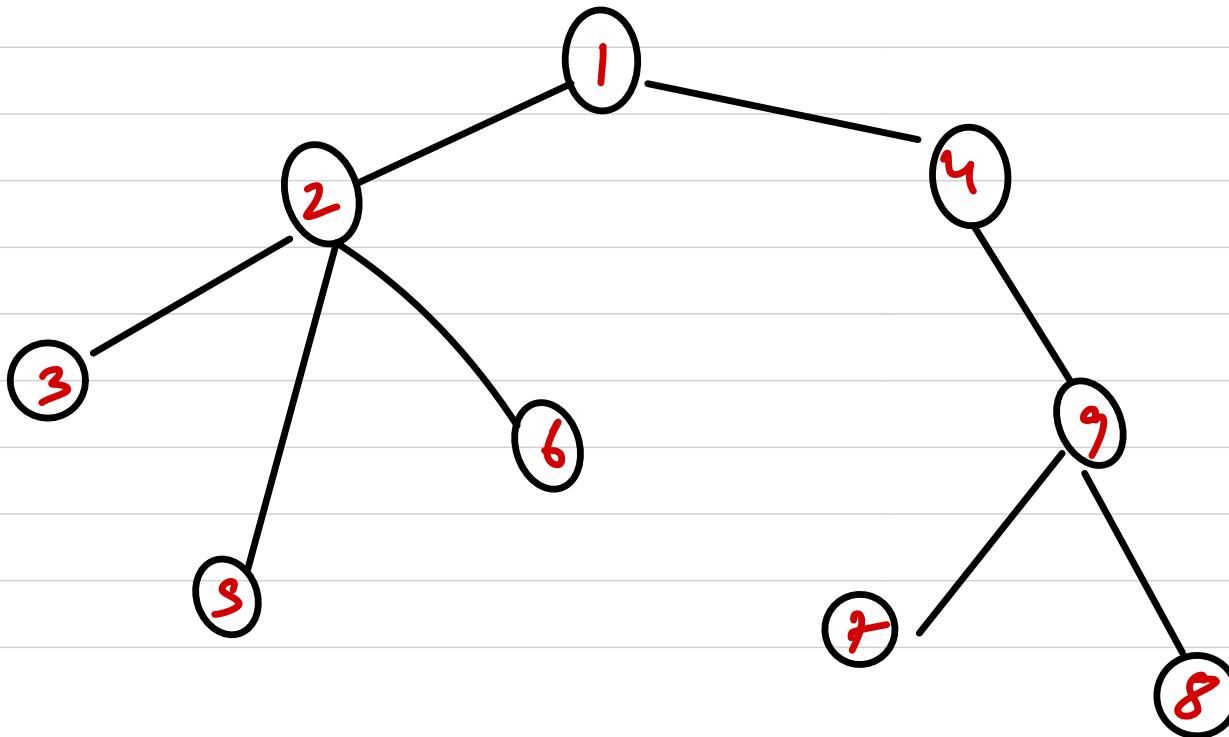
$$f(\text{node}, 0) = \pi \left( f(c_1, 0) + f(c_1, 1) \right)$$

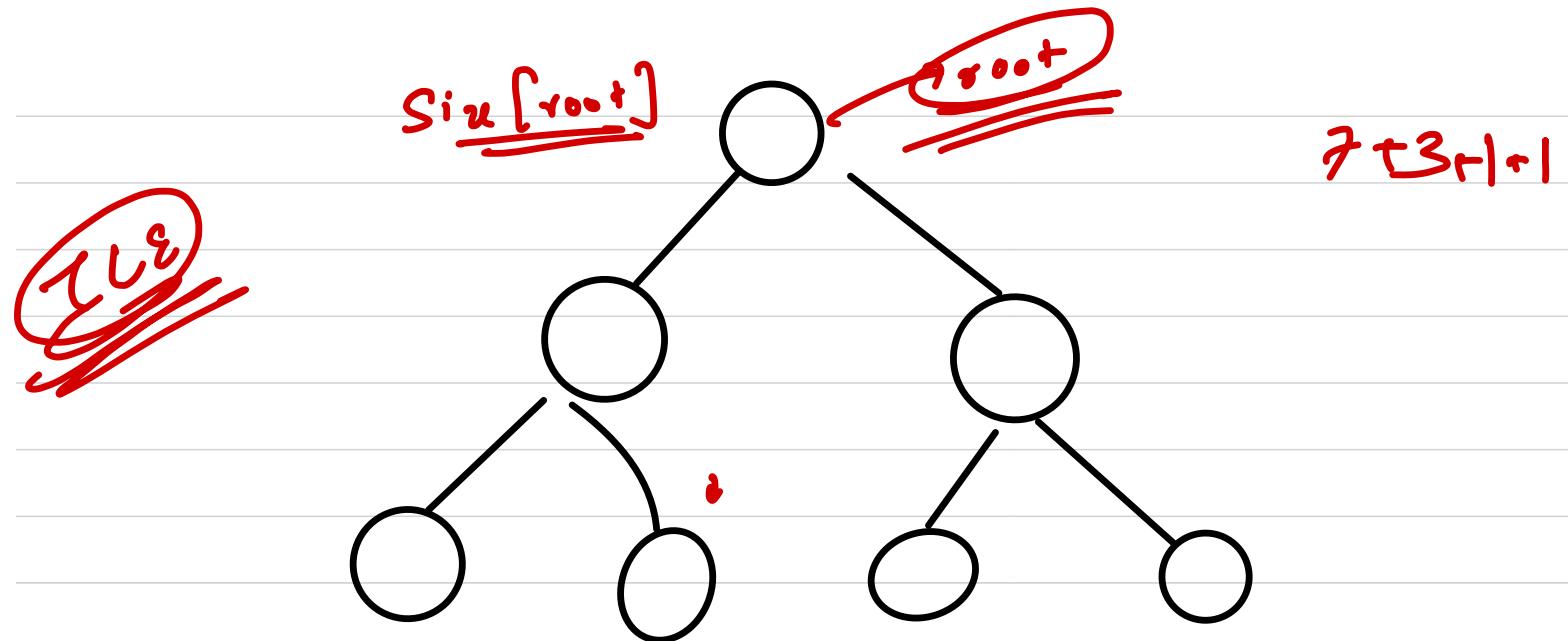
$c_i \in \text{child}$

$$f(\text{node}, 1) = \sum f(c_1, 1) \times \frac{f(\text{node}, 0)}{f(c_1, 0) + f(c_1, 1)}$$

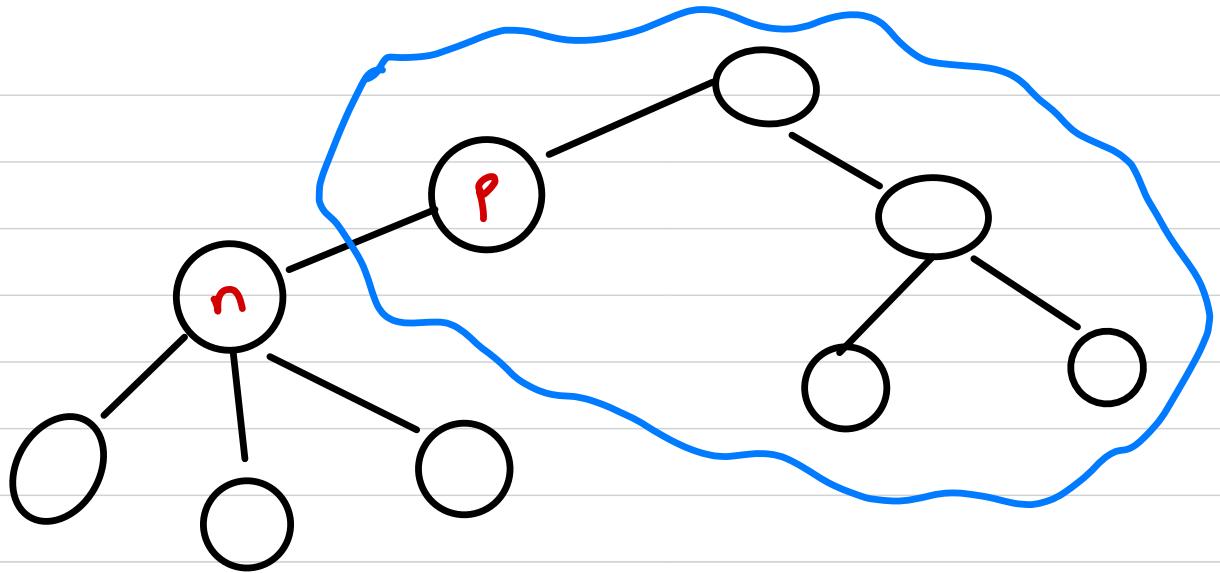


# Tree painting - ~~Codeforces~~

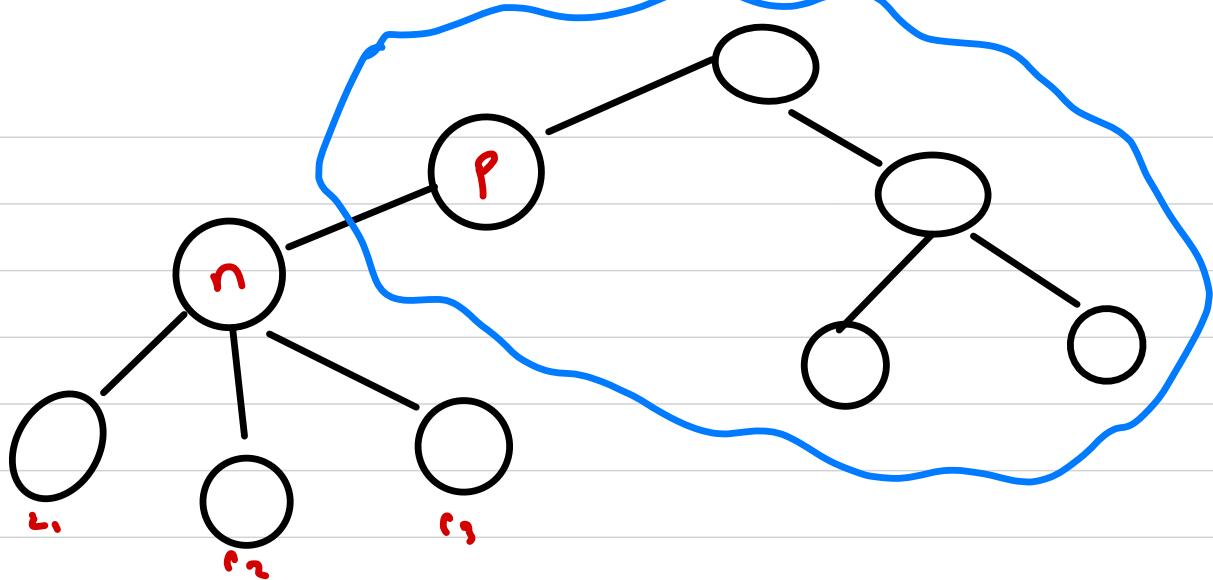




$$f(\text{root}) = \sum_{c_i \in \text{children}} f(c_i) + \text{Subtree } [\text{root}] + \underline{\underline{1}}$$

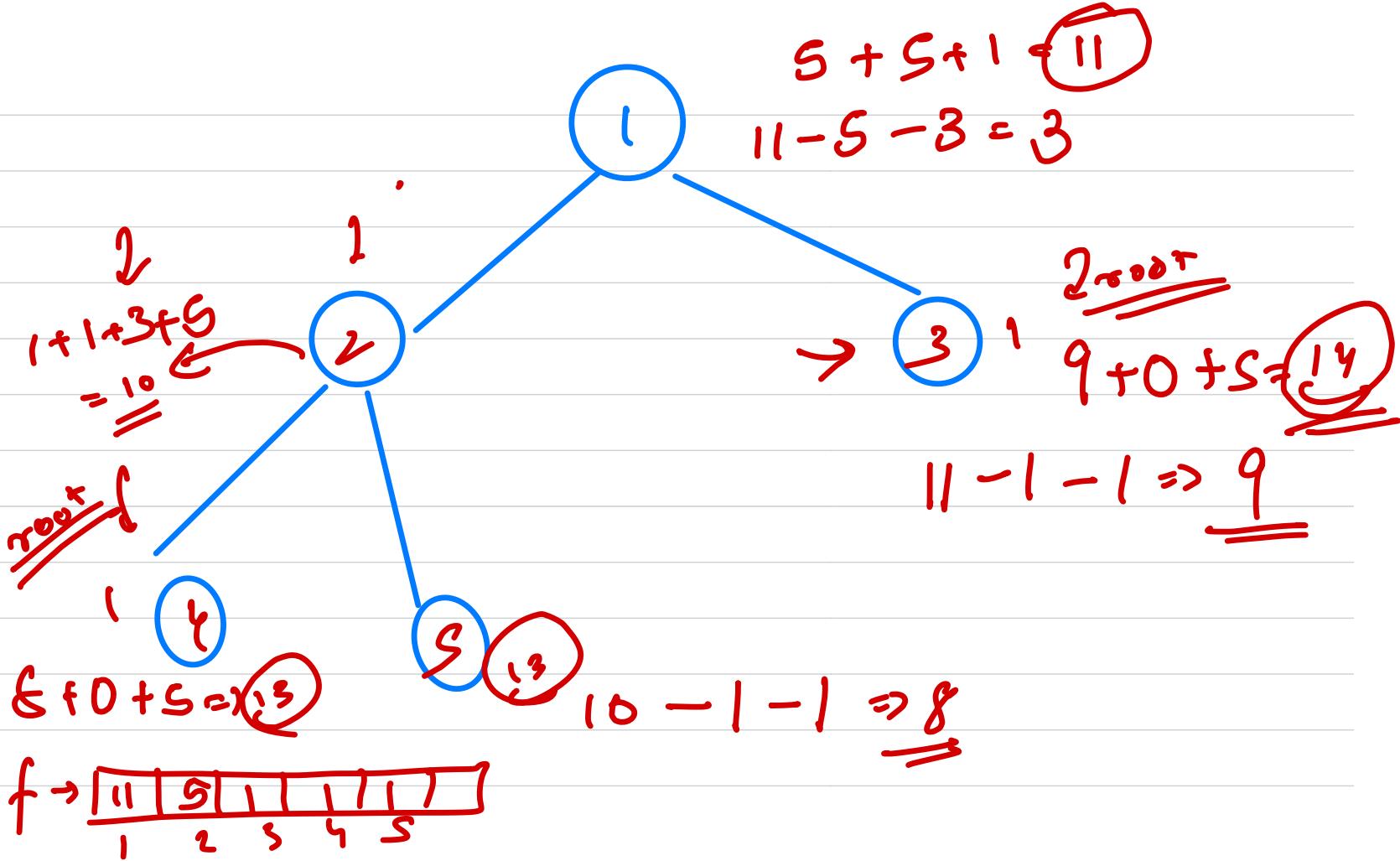


$g(p) \rightarrow$  contribution of any node  $p$  as an inverted  
parent.  
 $h(p) \rightarrow$  size of this inverted parent subtree

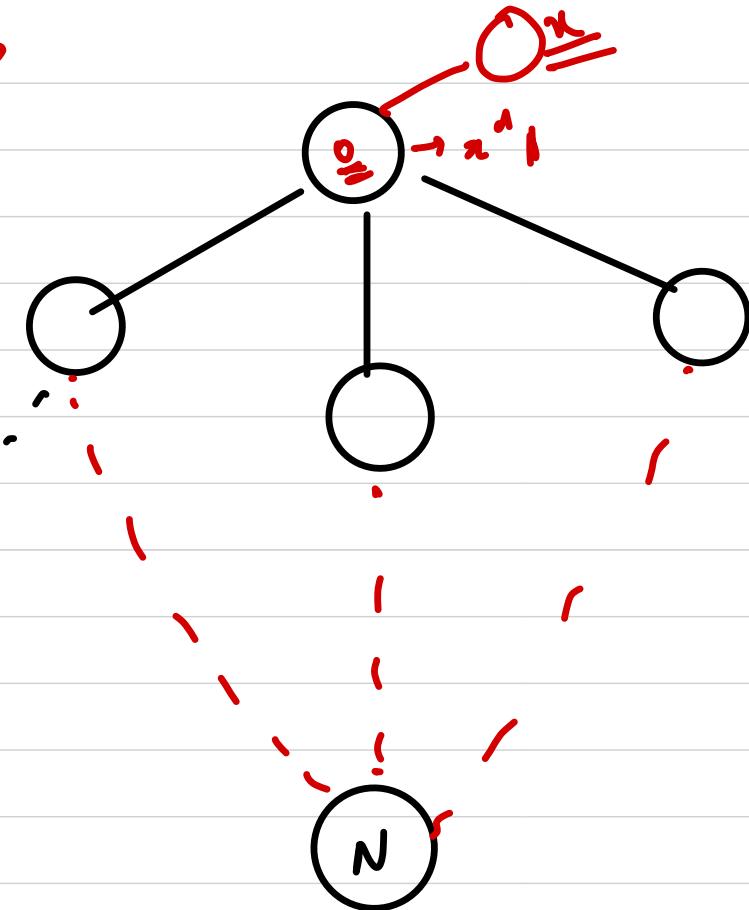


$$f(n) = \sum f(c_i) + g(p) + \text{subtree} + 1$$

$$g(n) = f(n) - f(\text{child}) - \underline{\underline{s_2(\text{child})}}$$



Xor circuit →



$$f(c, x) = \sum_{i=0}^3 (f(c_i, x^i))$$

$$\text{node} = N \rightarrow \underline{\underline{0}}$$

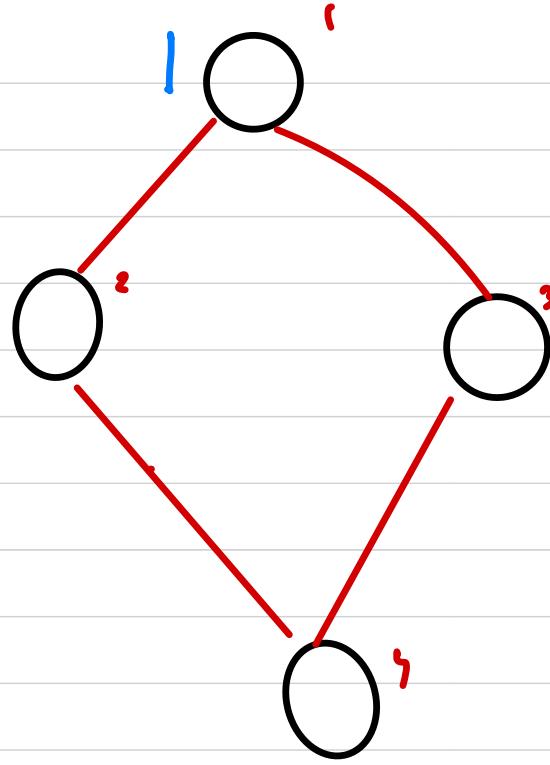
$$df(n, u)$$

$$u \leftarrow (0, \underline{\underline{3}})$$

$$1^2 = 3$$

$$1^3 = 2$$

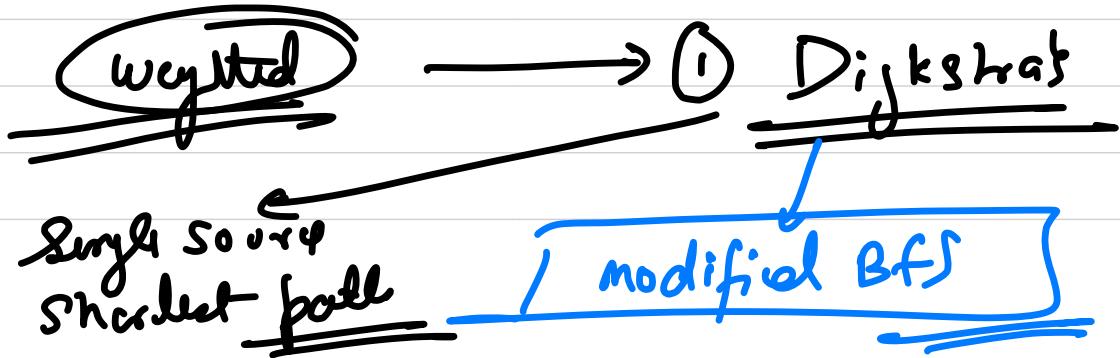
$$2^3 = 1$$



# Shortest path algorithms

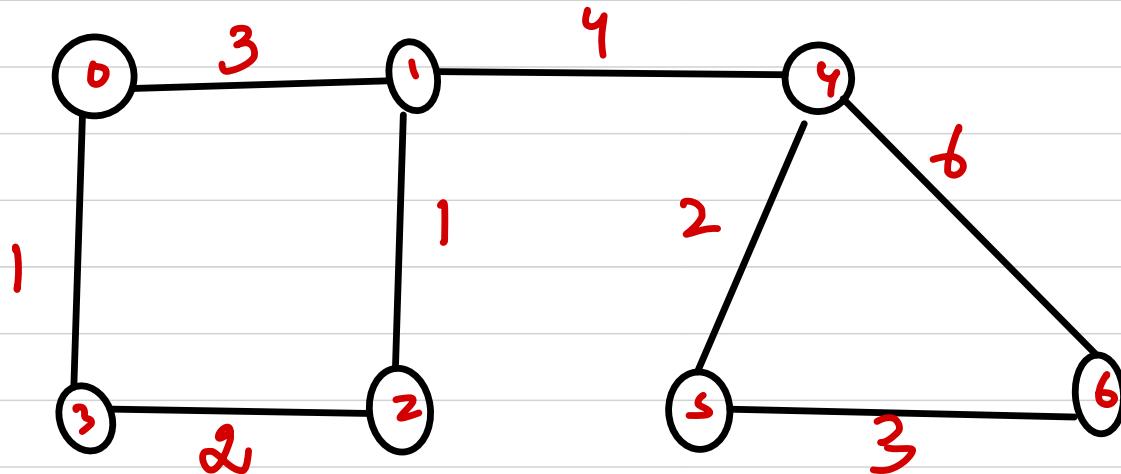


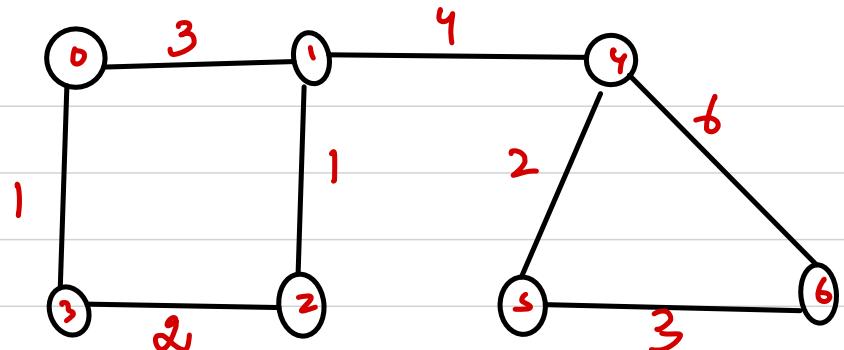
BFS → unweighted / uniformly weighted



→ greedy algo + Bfs → Dijkstra's

~~Src~~ ↗ 0





cviq

src → 0

Visited → Reached Sch

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

M <sup>in</sup> node	SD
x 0	0
x 1	3
x 2	3
x 3	1
x 4	7
x 5	9
x 6	12

if ( $\text{dist}[\text{src-node}] > \text{dist}[\text{src-par}] + \text{dist}[\text{par-node}]$ )

$\text{dist}[\text{src-node}] = \text{dist}[\text{src-par}] + \text{dist}[\text{par-node}]$

current SP from src to the node

#  $D(S, u)$ : min dist computed by Dijkstra's algo between  $S$  and  $u$ .

#  $y(S, u)$ : actual min dist between  $S$  &  $u$ .

→ propose →  $D(S, u) = \underline{\underline{y(S, u)}}$   
when  $u$  is marked visted or  
 $u$  is in the reached set.

Proof → Let's contradict.

Suppose the prev statement is false:

Then there are some vertices when a vertex  $u$  is included in the visited/reached set :

$$D(s, u) > y(s, u)$$

Let  $x$  is the first vertex included in visited/reached set.

if  $x$  is the first vertex :

$$D(S, x) > g(S, x)$$

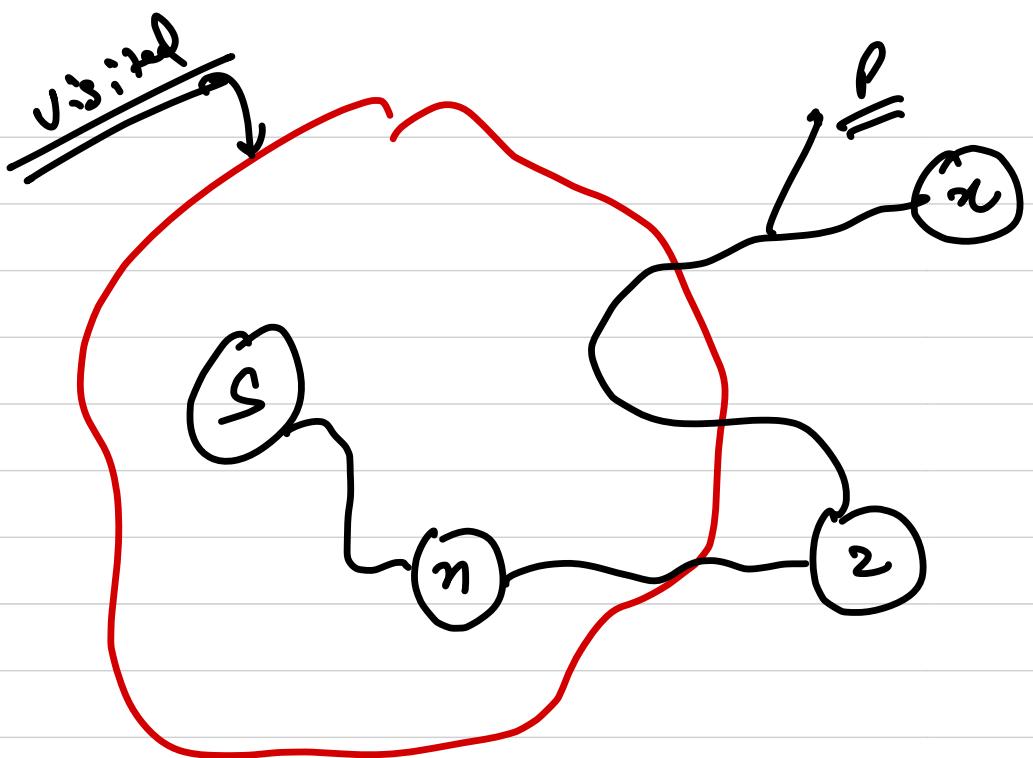
this implies all prev vertices  $m$ , that were included in visited/reached set follows

$$D(S, m) = g(S, m)$$

let's analyze the moment when  $x$  is going to be included.

Let's say,  $P$  be the real shortest path from  $S$  to  $x$ .  
Let's say,  $z$  is the first vertex not in the visited set and is on the shortest path.

Let's say  $n$  is the predecessor of  $z$  included in the path  $P$ . and included in visited.



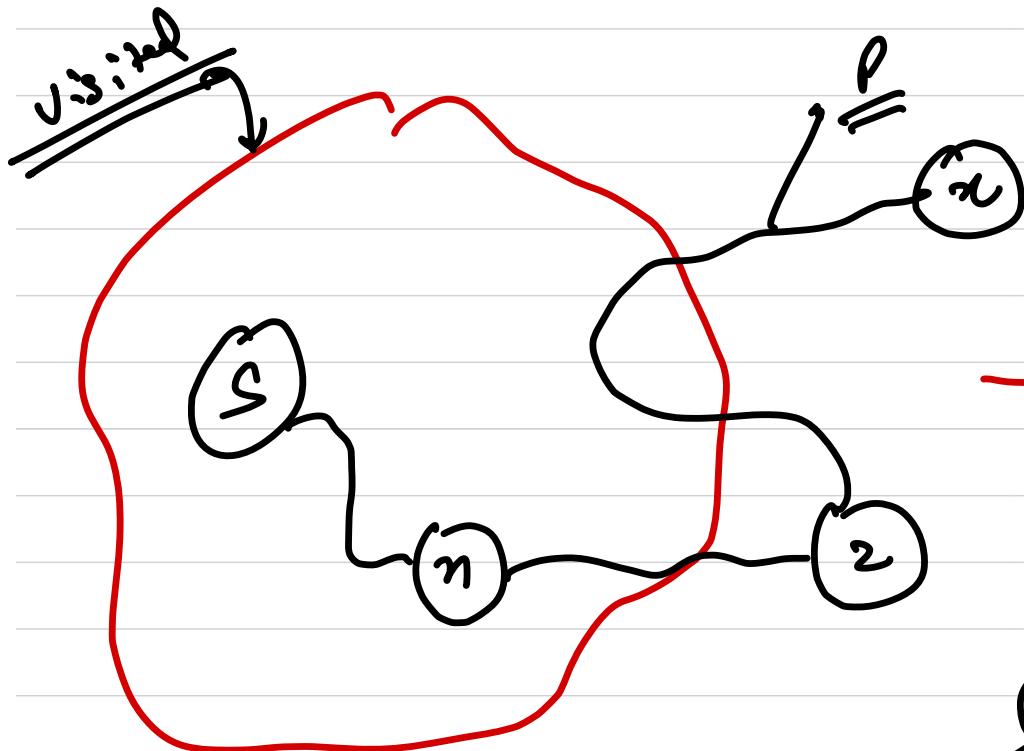
non visited

We say

$$D(S, n) = \gamma(S, n) - \textcircled{1}$$

$$\begin{aligned} D(S, z) &= D(S, n) + \text{edge cost}(n, z) \\ \gamma &= \gamma(S, n) + \text{edge cost}(n, z) \end{aligned}$$

$$D(s, x) \leq D(s, z) \leftarrow$$



$$y(s, x) =$$

$$y(s, z) + y(z, x)$$

$$\underline{D(s, x)}$$

$$D(s, z)$$

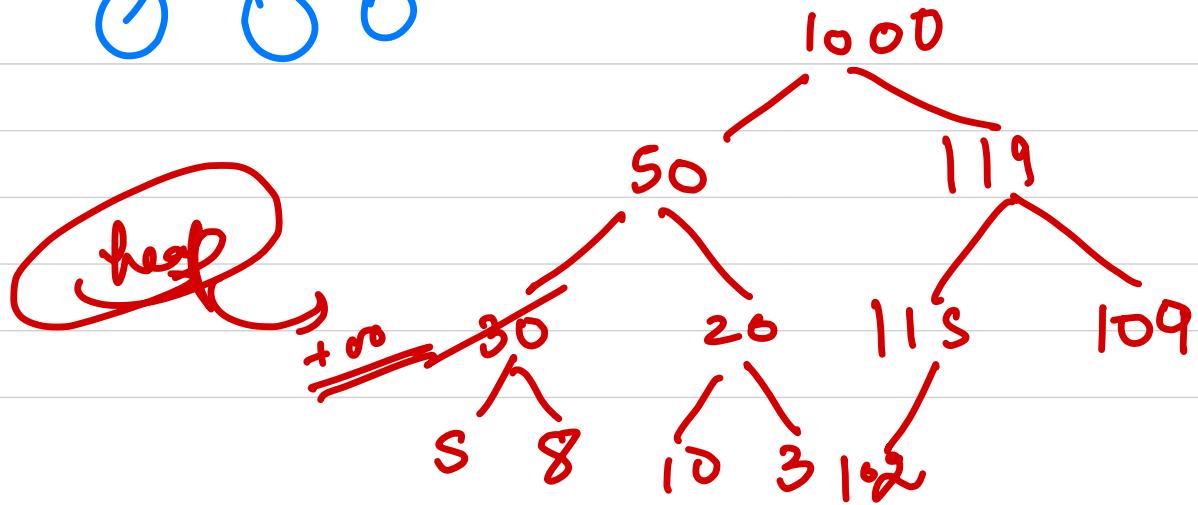
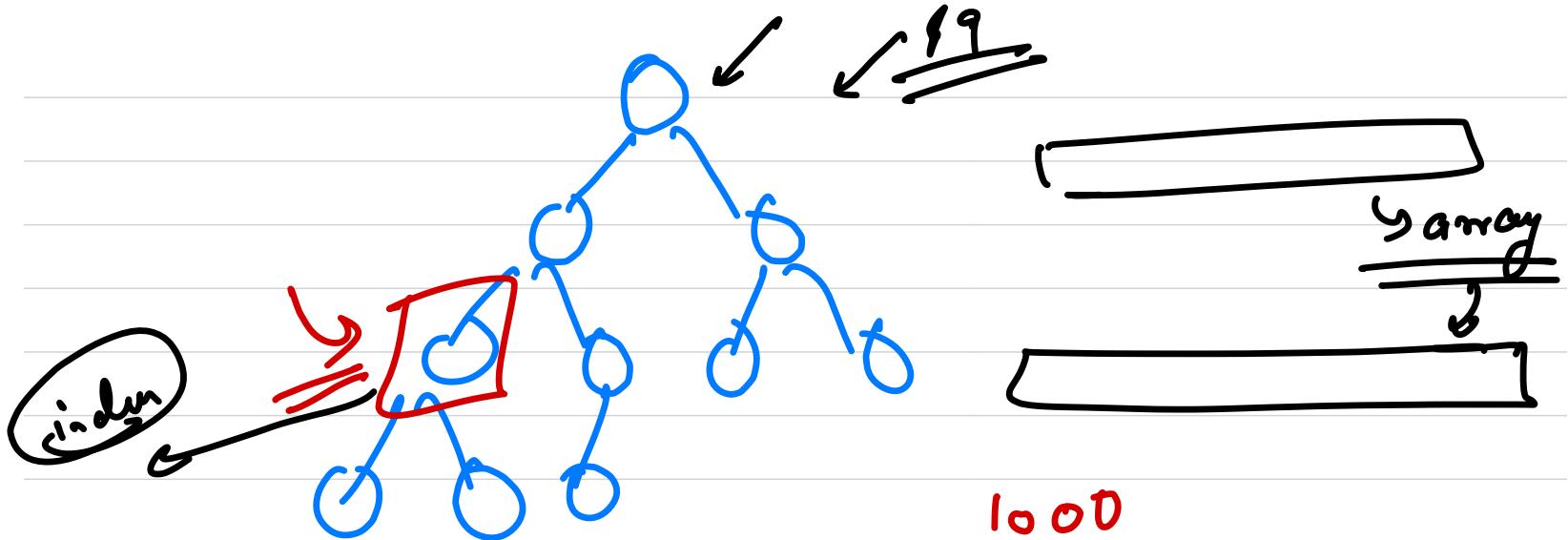
$$D(s, z) \leq D(s, n)$$

$$D(s, n) \leq g(s, n) + cost(n, z)$$

and  $D(s, z) \leq g(s, n) + cost(n, z) + g(z, z)$

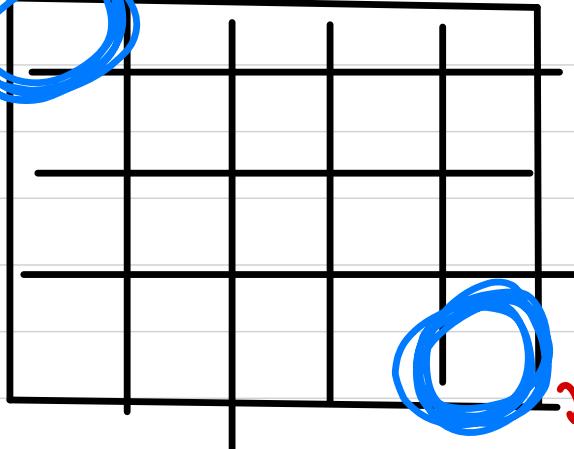
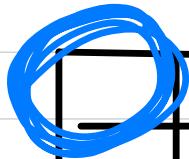
$$D(s, n) \leq \underbrace{g(s, n) + cost(n, z)}_{\text{red bracket}} + \underbrace{g(z, z)}_{\text{red bracket}}$$

$$D(s, n) \leq g(s, n)$$



~~grid~~

~~grid~~  
~~grid~~



Dijkstra

grid graph

$g[i:j, j]$  → cost of  
travelly through a

cell  $i, j$

gen → no ← every cycle

$i, j \rightarrow$

-cell →  
↓  
→

$$f(n) = f(n-1) + f(n+1)$$

trapxx

Dijkstra



$$\begin{aligned} f(i,j) &= f(i,j-1) \quad f(i-1,j) \\ &\quad f(i+1,j) \quad f(i,j+1) \end{aligned}$$

~~P~~  $\Rightarrow$  N cities are there, and  $i^{\text{th}}$  city is at height  
from base sea level. You have a truck that requires  
no fuel but can go only in either "up" or "down". These  
cities are connected by K bidirectional roads. You want to  
travel from City 1 to N with min cost because, when  
you change mode of the truck you need to bear  $C_i$  cost  
if you are at  $i^{\text{th}}$  city currently. if impossible point =

$N \rightarrow 6$

$K = 6$

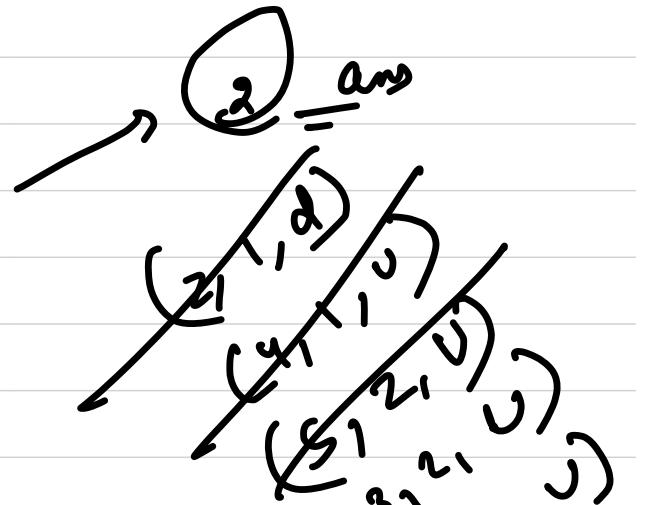
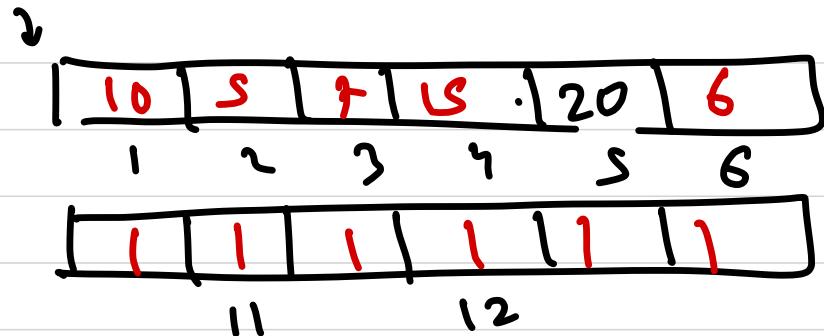
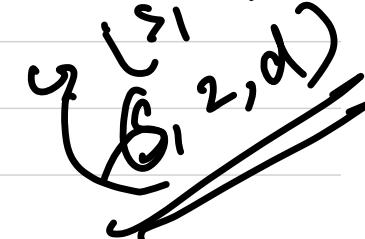
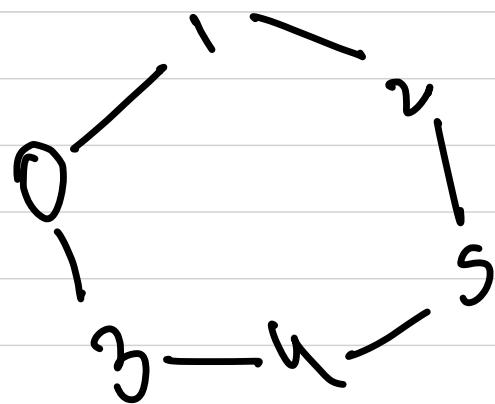


Table showing values for indices 1 through 6:

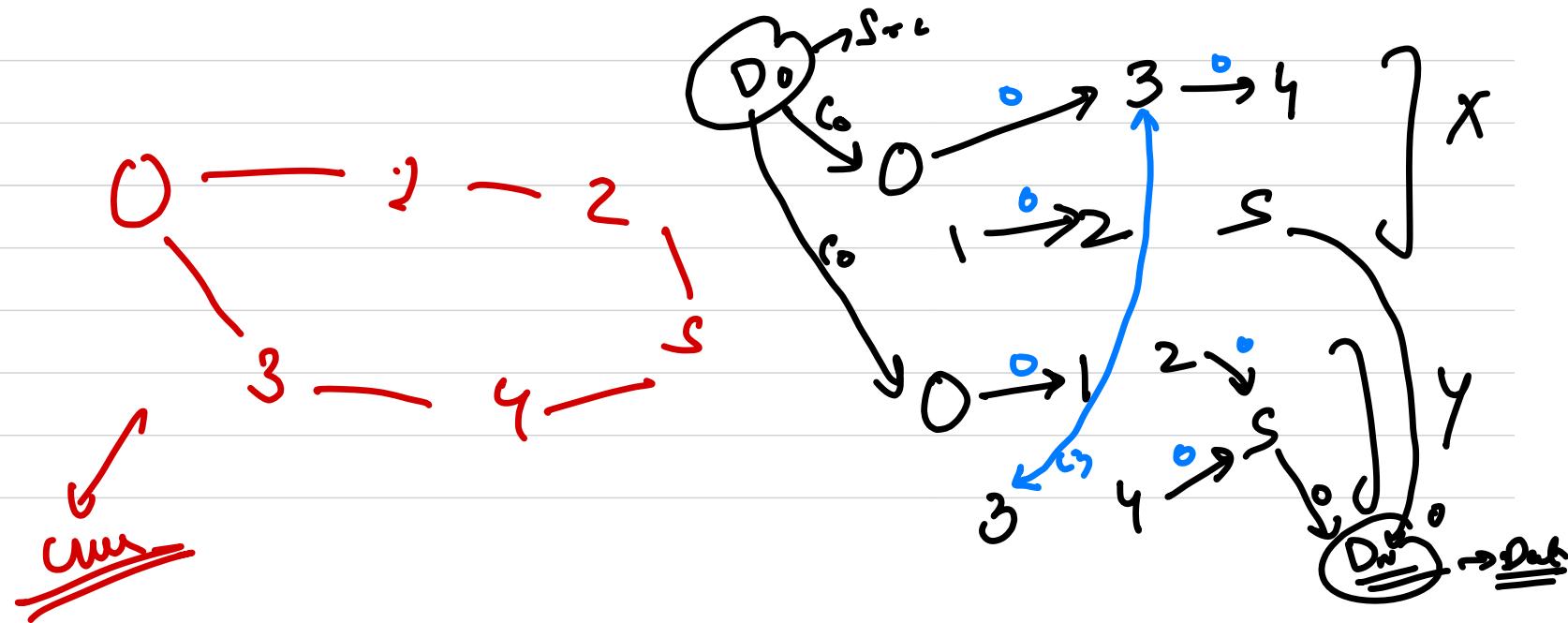
1 - 1
2 - ∞
3 - ∞
4 - ∞
5 - ∞
6 - ∞

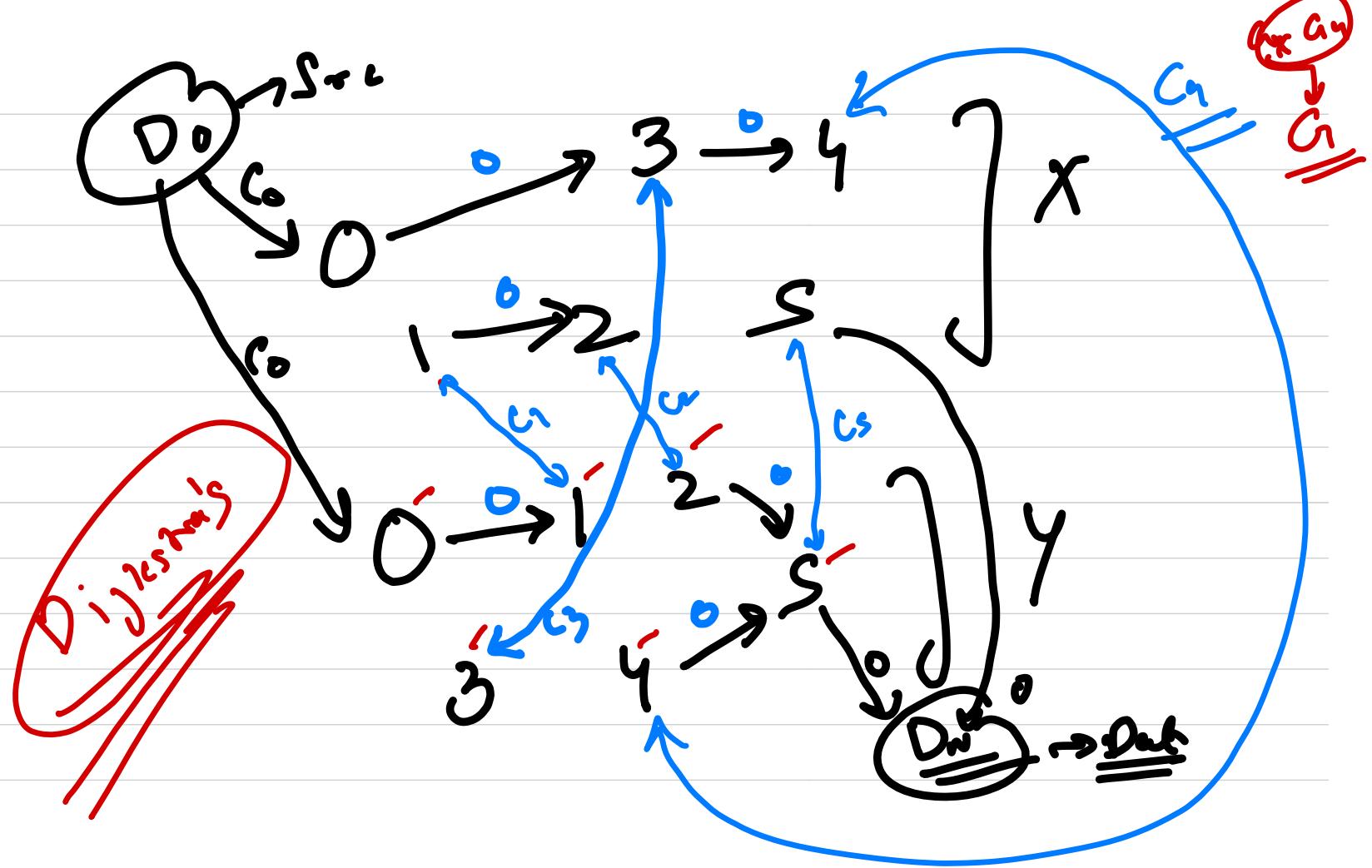


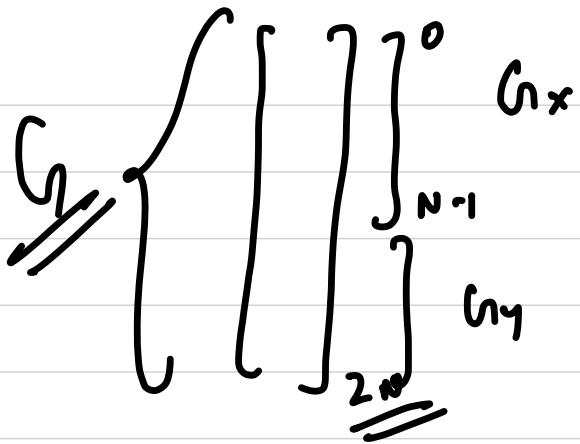
10 S 7 15 20 6

1 1 1 1 1 1  
0 1 2 3 4 5

$h_1 \geq h_2$

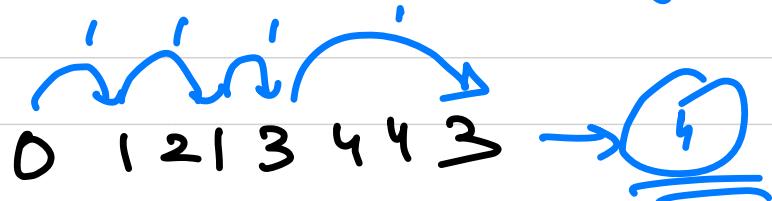






Q: Given a string of digits of length  $N$ .  
 $(N \leq 10^5)$ . You're standing at  $0^{th}$  index.  
We want to reach at  $(N-1)^{th}$  index.

For each jump, you take a cost of 1 unit &  
from any  $i^{th}$  digit, you can go to  $i-1$  or  $i+1$   
or any other digit  $S_j$  such that  $S_i = S_j$ .  
Find min jumps.



$[0, 9] \rightarrow \underline{\text{digits}}$

0 1 2 1 3 4 4 3

worst case  
6 digit  $\rightarrow$  twice

$T_{\max} \rightarrow 19$

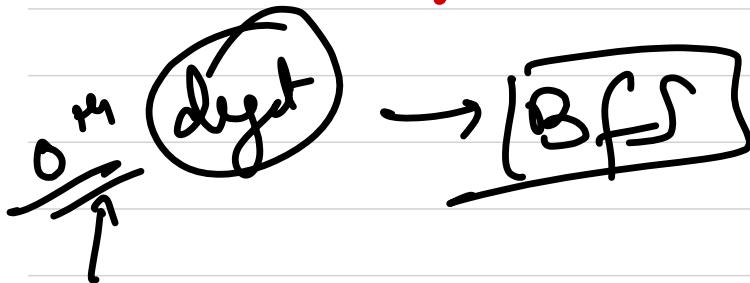
$O(n)$

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

dist 1 2 3 4 3 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

$\rightarrow$  90  $\rightarrow$  1, 2, 3, 4, 2, 3, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9

digit → occ → i, j, k, l



cur ← q.pop



- first

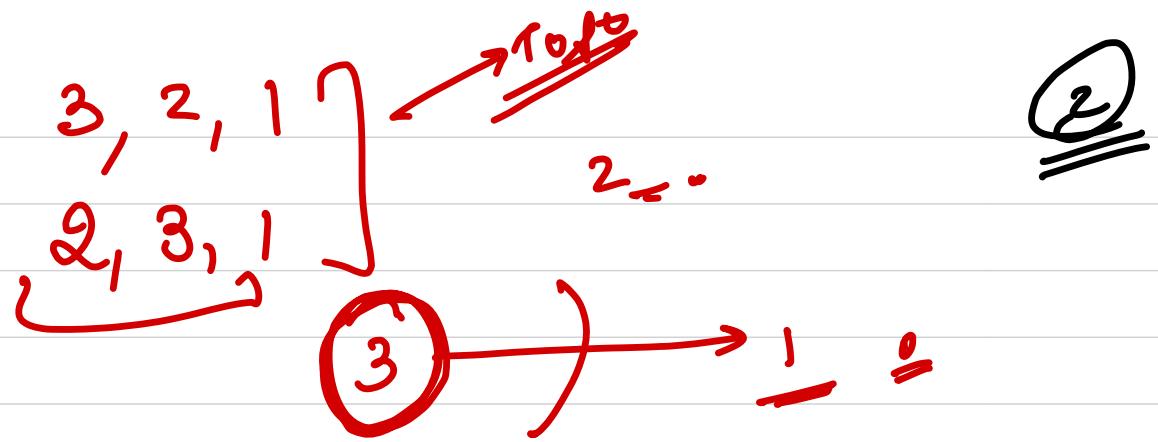
~~Q:~~ You've a directed acyclic graph, & you want to add few edges to this DAG such that after addition it remains a DAG.

find the max edges you can add. Point in lexicographic order

$$N \leq 1500$$



A sequence of edge is lexicographically smaller than other based on the first edge differing in the given sequence



$\begin{matrix} 2, 3 \\ 2, 1 \\ 3, 1 \end{matrix} -$

$(w, v)$

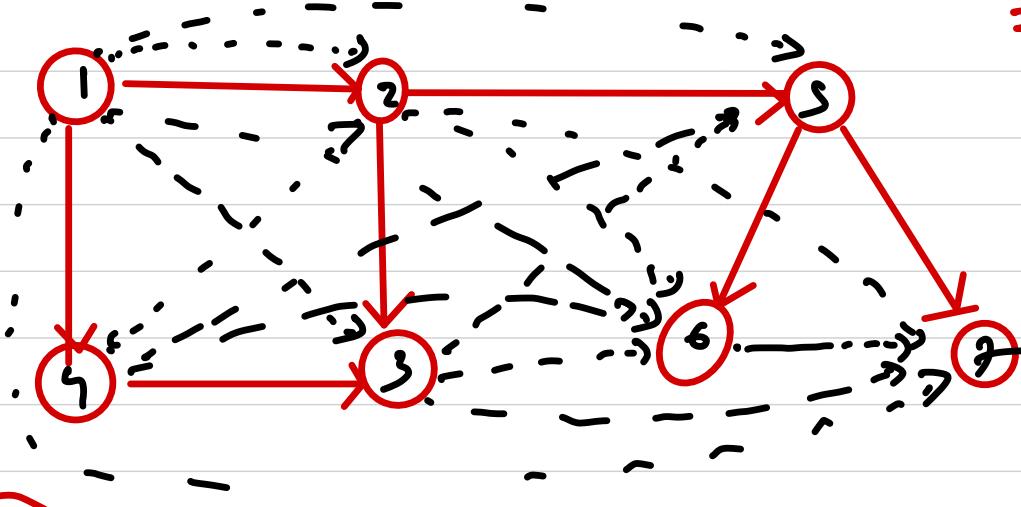
$e_1 < e_2$

if  $e_1 \cdot v < e_2 \cdot v$

$\begin{matrix} 1, 2 \\ 3, 2 \\ 3, 1 \end{matrix} -$



same



$U \rightarrow U$   $\rightarrow$  ~~gen~~  
~~by~~

$V \rightarrow V$   $\times \times V$

$\rightarrow 1, 4, 2, 3, 5, 6, 7 \rightarrow \underline{\text{Topo order}}$

$(U, v)$

$U < V$

then we can add an edge  
from  $U \rightarrow V$  if it doesn't exist

$E \leftarrow \underline{\text{maximal set}}$

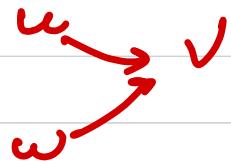
$|E| \rightarrow \underline{\text{Topo}} |E'| \geq |E|$

outflow  $\rightarrow 0$   $\rightarrow$  (max magnetized)  $\rightarrow$  min

$$(u \leftarrow v)$$

$$\begin{aligned} U_0 &= \pm 0 \\ V_0 &= \pm 0 \end{aligned}$$

$$u < \omega < v$$



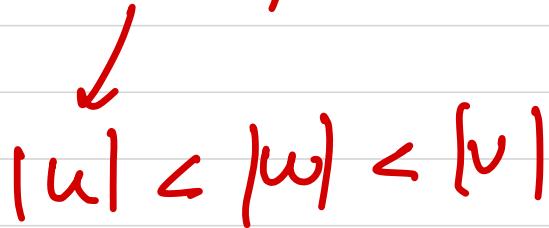
$$\underline{\underline{\omega}} \rightarrow \underline{\omega'}$$

$$v \rightarrow \omega$$

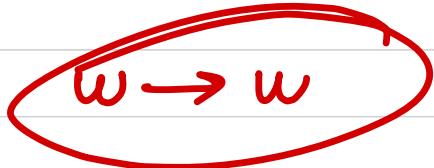
$$|\omega| < |v| \leftarrow$$

outside  $\rightarrow v$

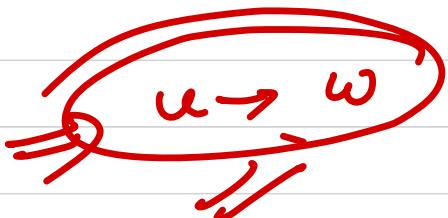


$$|u| < |\omega| < |v|$$


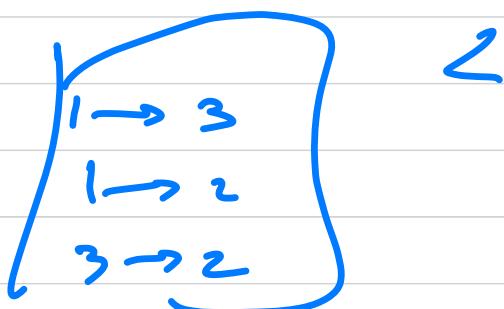
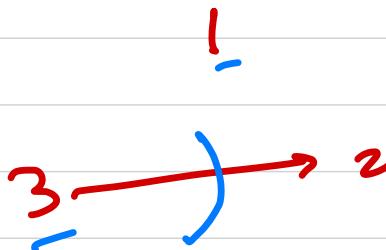
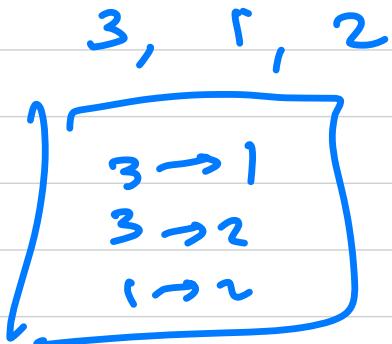
$$u \xrightarrow{\omega} v$$


$$\omega \rightarrow w$$


⋮  
-

$$u \rightarrow w$$


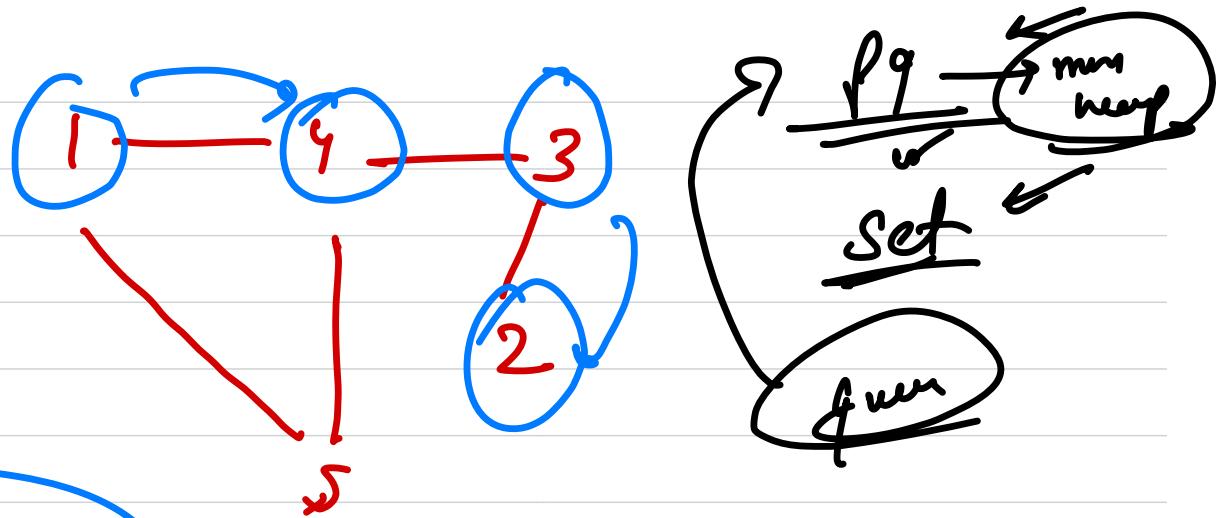
$\rightarrow \underline{\text{index}}$



1, 3, 2



$O(f_{n+m})$



1, 4, 3, 2, 5

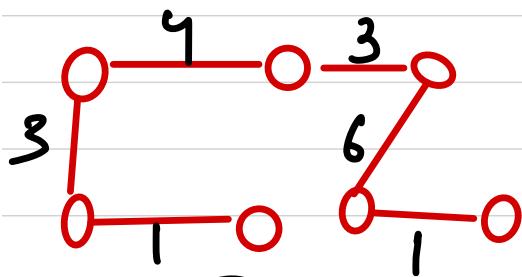
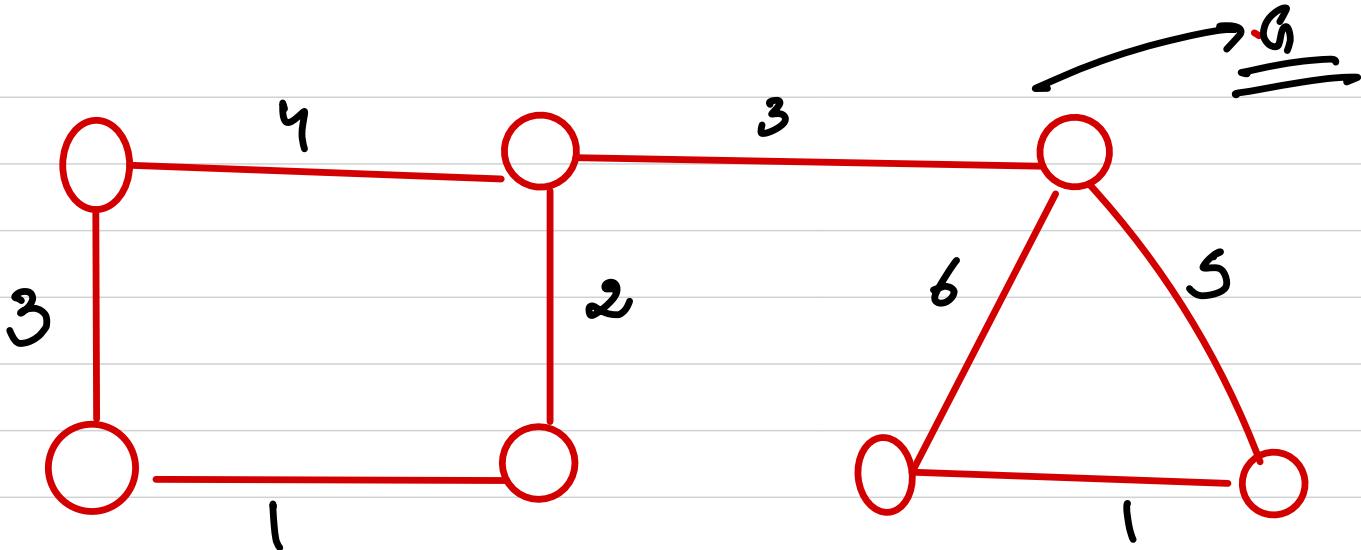
## Minimum Spanning Trees

Tree → graph without cycles

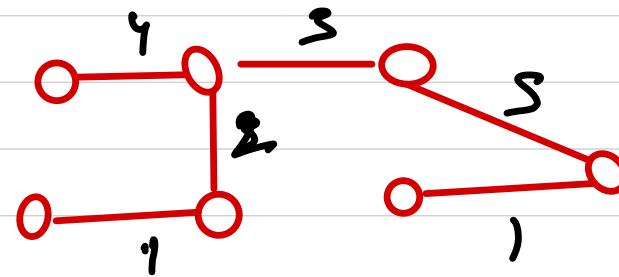
Spanning Tree → It is a subgraph that is a tree and includes all vertices and is always

connected.

MST → spanning tree with minimum total  
edge weight:



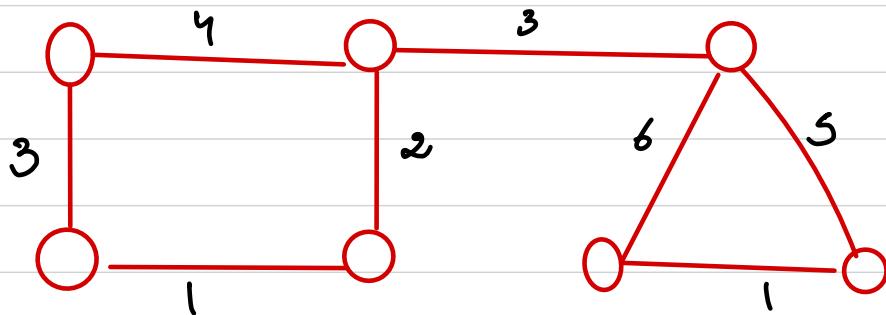
18



16

greedy

Kruskal's Algorithm



Avoid cycles

Cycle detection

edges pick → start from min wt edge

such that it doesn't

form cycle.

DFS

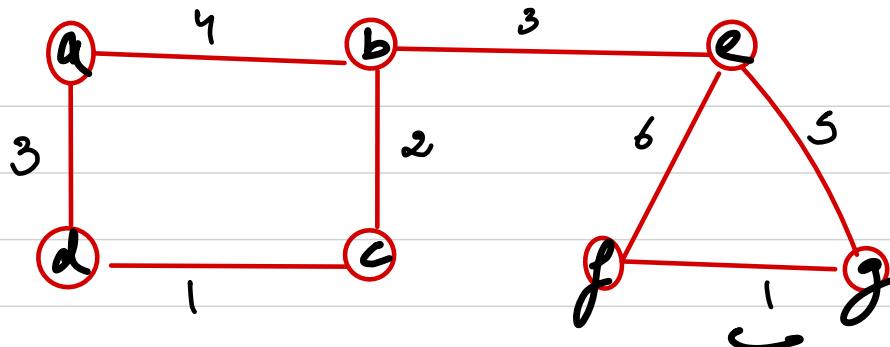
DSU

Standard

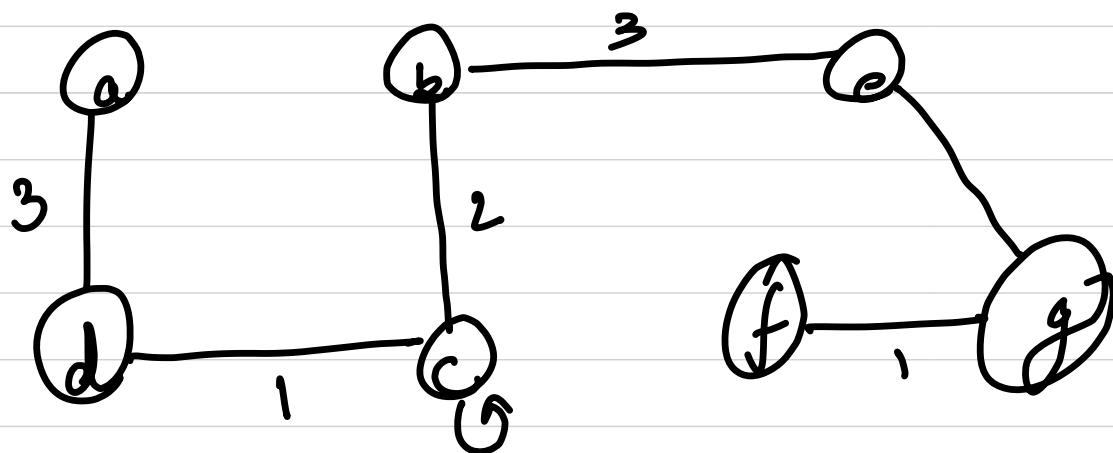
$$\Sigma = [e_1 \ e_2 \ e_3 \ \dots \ \dots \ e_{n-1}] \quad \leftarrow$$

$\xrightarrow{\text{is } \cong}$

$$\Sigma' = (e'_1 \ e'_2 \ e'_3 \ \dots \ \dots \ e'_{n-1}) \quad \leftarrow$$



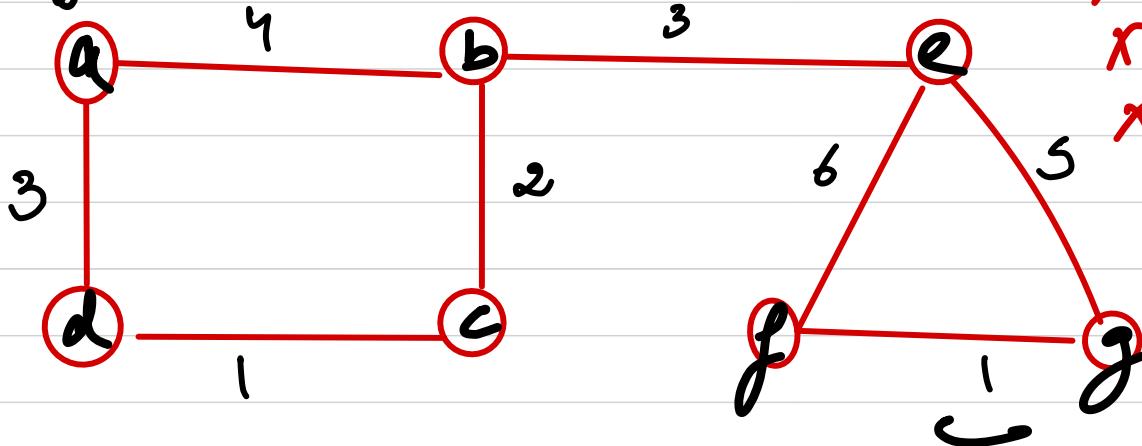
color sort()



$E \log E f$   
 $[E \log^* V]$   
 $O(\underline{\underline{E \log E}})$

*goes* *to*

*scc*



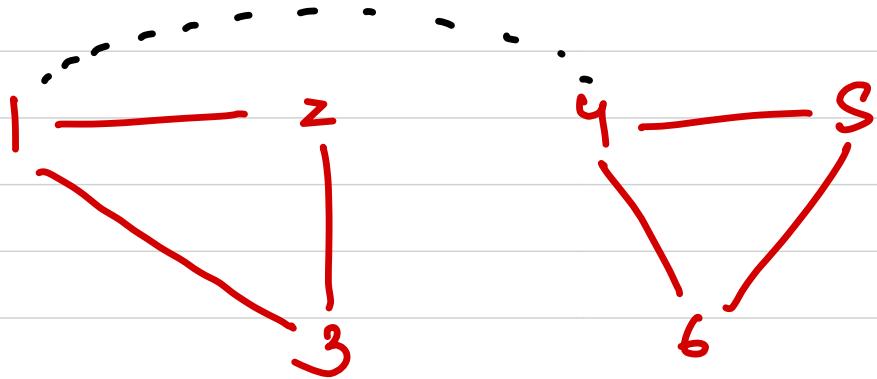
x	a	p
x	0	~1
x	b	2
x	c	1
x	d	3
x	e	3
x	f	1
x	g	9
s	c	5
c		L

$O((V+E) \times \log V)$

~~Qn~~ Given a graph of N nodes, each node has a cost associated. There are some edges present & we need to add few more to connect this graph. We can add an edge b/w nodes u & v if  $\text{cost}_u \geq 0$  &  $\text{cost}_v \geq 0$ .

Construction of edge costs  $\rightarrow \text{cost}_{uv} + \text{cost}_{vu}$ .

Find min cost to connect the graph.



node cost

1 - 1

2 - 3

3 - 5

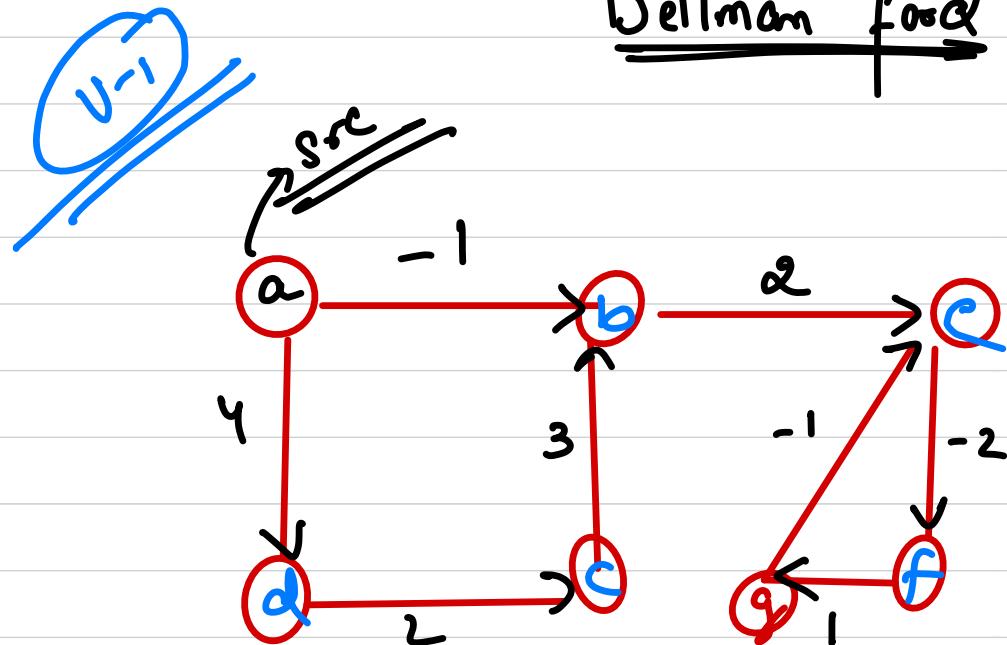
4 - 2

5 - 4

6 - 6

↓  
③ ans

Bellman Ford  $\rightarrow \underline{\underline{PF}}$   
 $\hookrightarrow \underline{\underline{SSSP}}$   $\equiv \underline{\underline{BV}}$



relaxation

$\text{if } (\text{SP}[u] > \text{SP}[v] + \Sigma_{v,u})$   
 $\text{SP}[u] = \text{SP}[v] + \Sigma_{v,u}$

a, b, c, d, e, f, g  
 0,  $\infty$ ,  $\infty$ ,  $\infty$ ,  $\infty$ ,  $\infty$ ,  $\infty$   
 7, 6, 9



for  $i \leq 1 - (N-1)$

for each edge  $(u, v)$   
relax  $(u, v)$

$f(x, y)$

↓  
length of shortest  
path whose leg  
is almost  $y$  and

path is from

src to  $x$

$w \rightarrow \text{edge wt}$

$$= \begin{cases} 0 & y == 0 \\ \infty & y \neq 0 \text{ and } x \neq \text{src} \\ \min(f(x, y-1), \min(f(u, y-1) + w(u, x))) & \text{else} \end{cases}$$

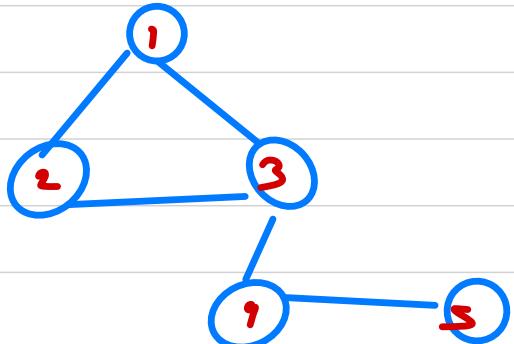
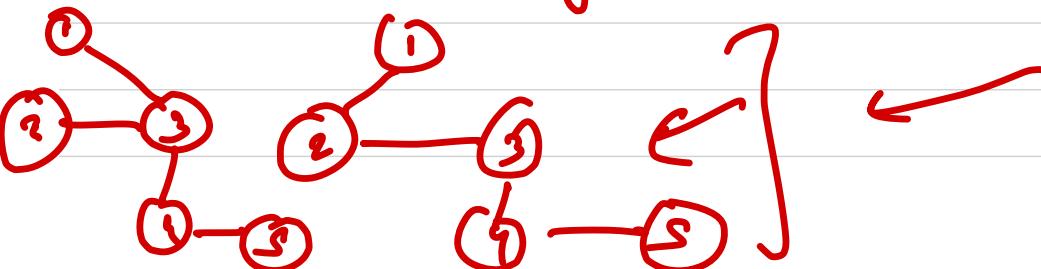
$y == 0$   
 $x == \text{src}$

$y \neq 0$   
 $x \neq \text{src}$

$(u, v) \in E$

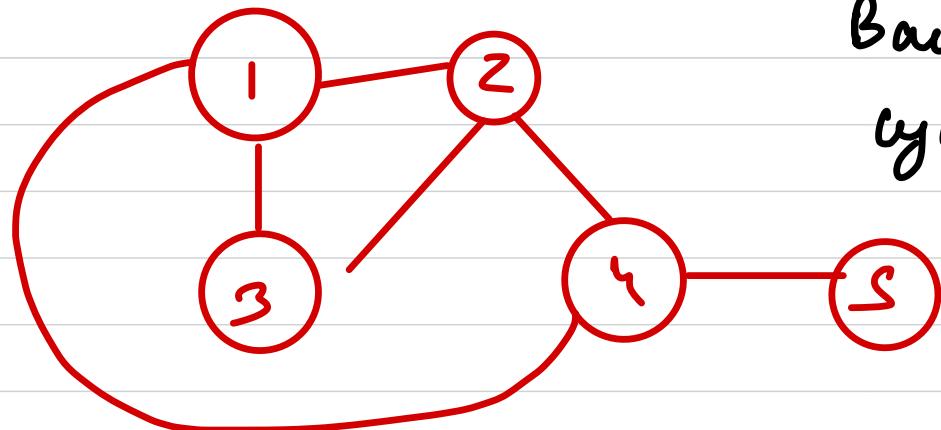
## Articulation Points & Bridges

\* DFS Tree → If we perform dfs on graph such that an already visited node is not visited again, then dfs forms a tree structure called as dfs tree.



There can be multiple dfs trees.

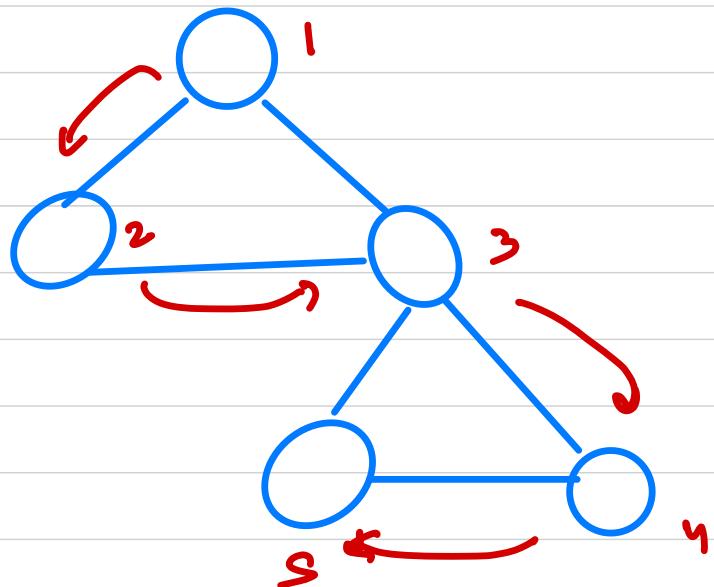
\* Back Edge → So this edge connects a node x to an already visited node in graph.



Back edge leads to cycle in dfs tree-

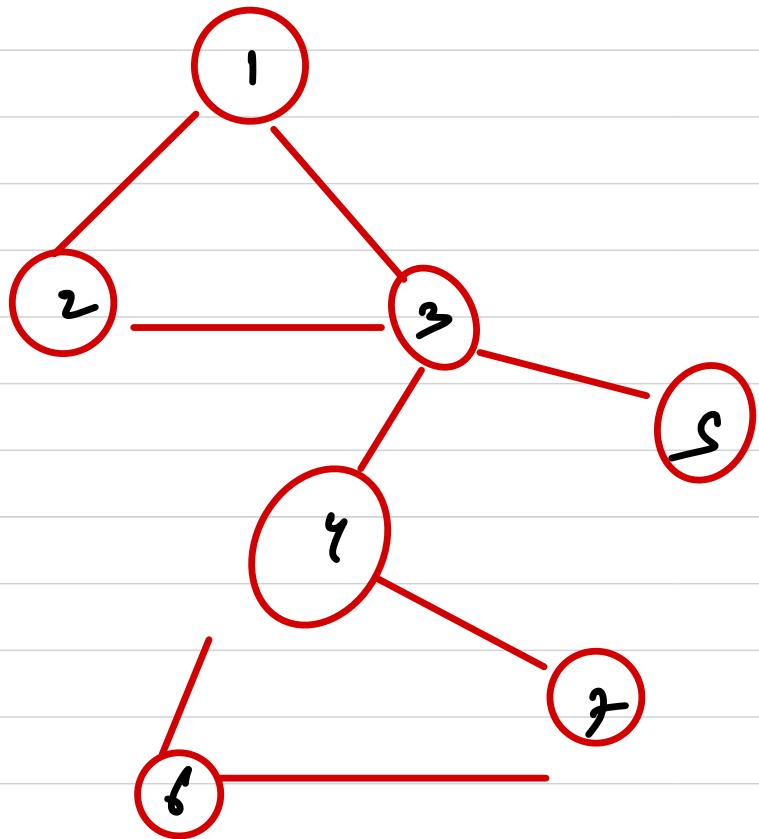
Span edge → edges included in dfs tree.

\* Discovery time instant  $\rightarrow$  the first moment in time when you meet a node in off.



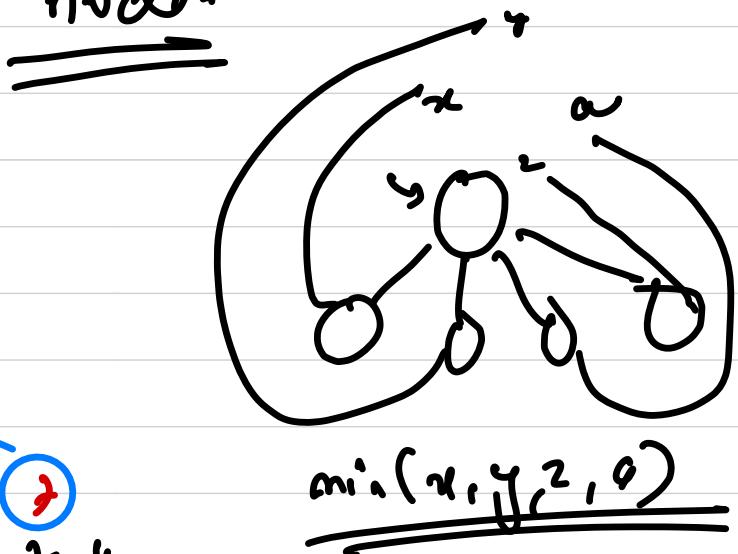
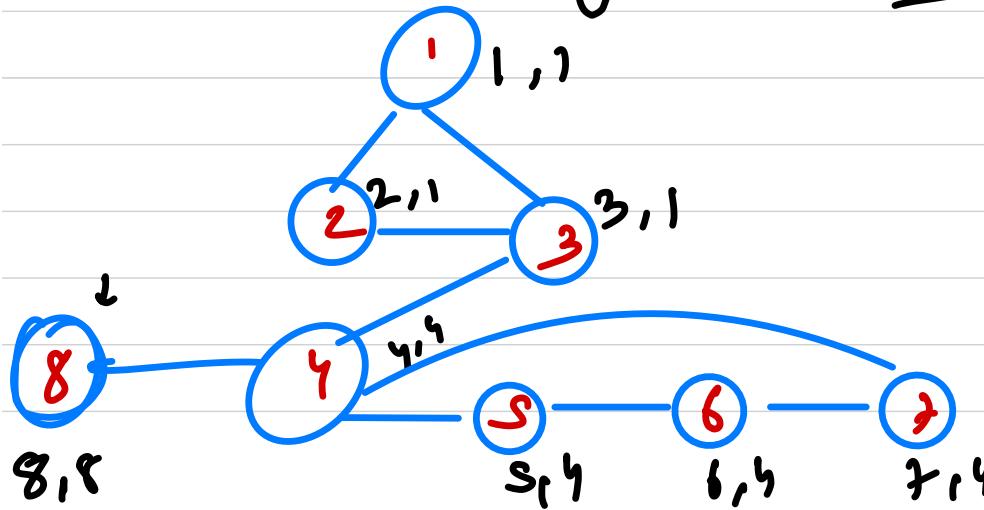
Articulation point  $\rightarrow$  A vertex node in a graph is called articulation point, if by removing the node & edges correspondingly to it, we split the graph into more components i.e. the no. of connected components increase.

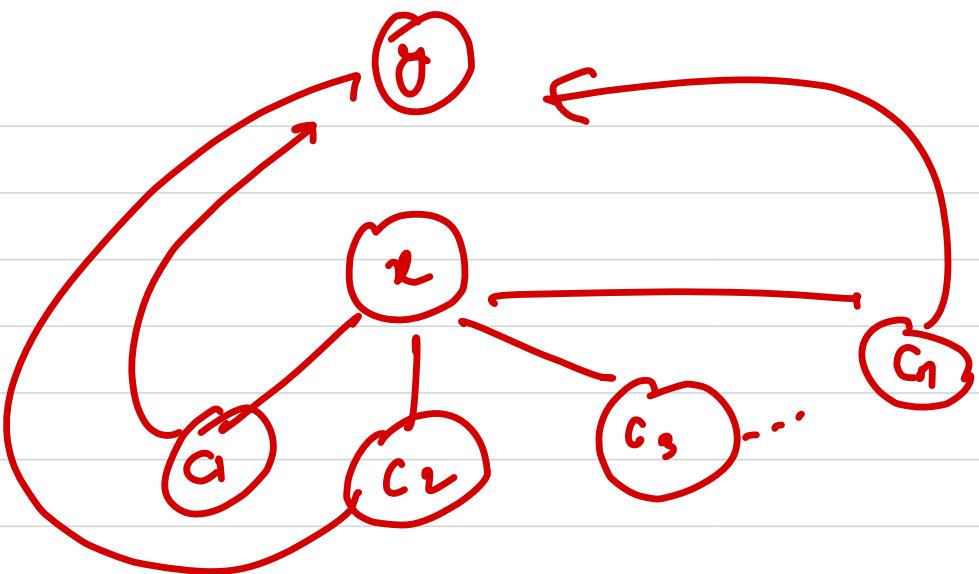
Bridge  $\rightarrow$  An edge is a bridge if by removing it we increase the no. of CC.



# Lowest time unit → minimum discovery time  
of a node that any back edge is pointing

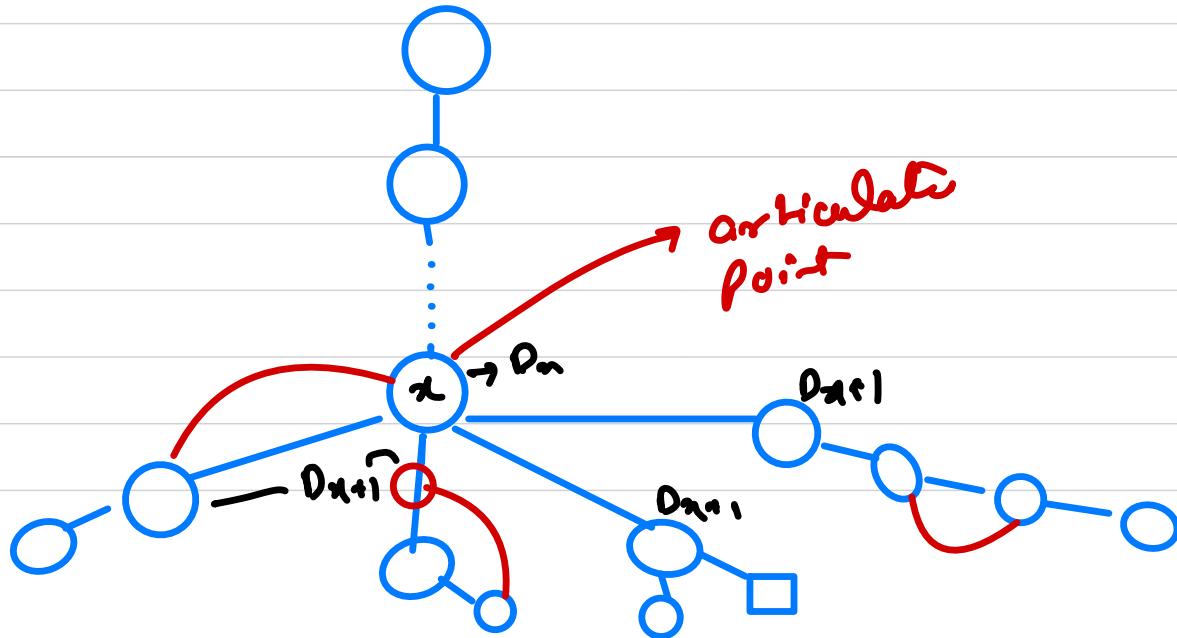
in the subtree of current node.

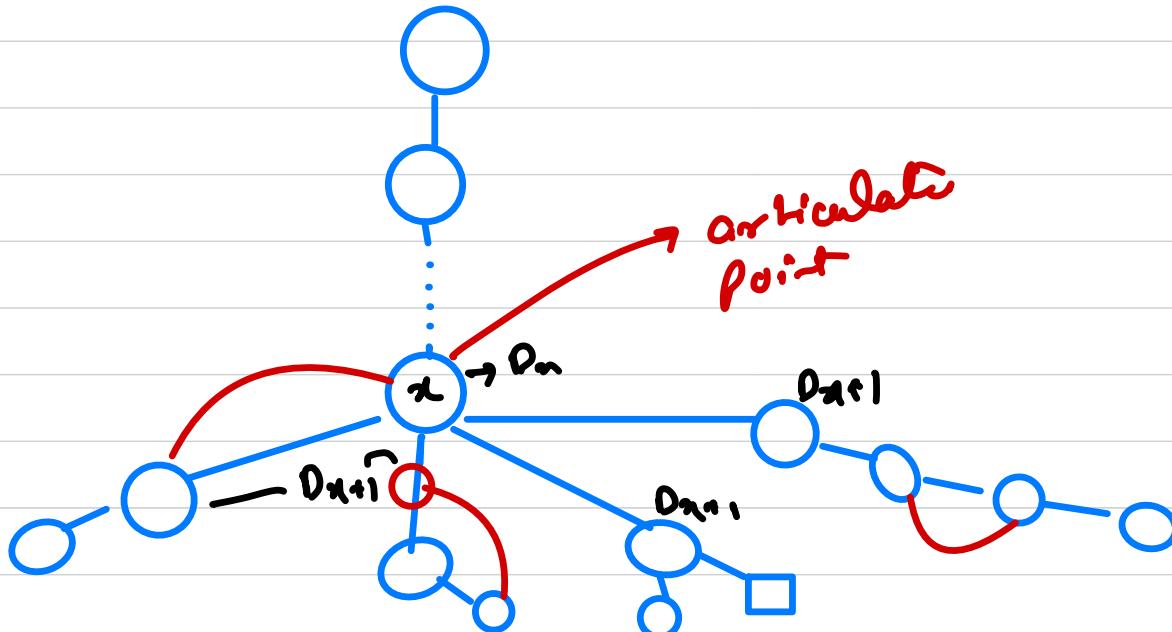




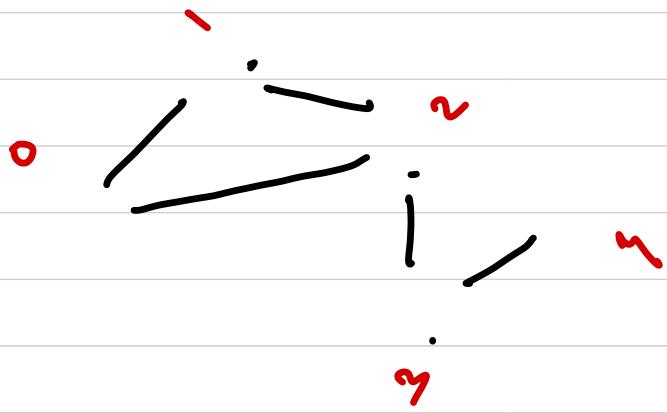
if any child of node  $x$ , if it has a back  
~~/ or lowest time~~  
 to a node, with discovery time greater or equal to  $x$   
 then  $x$  is an articulation point

If a node  $x$  has any child with lowest time more or equal to discovery of  $x$ , then  $x$  is an articulation point.



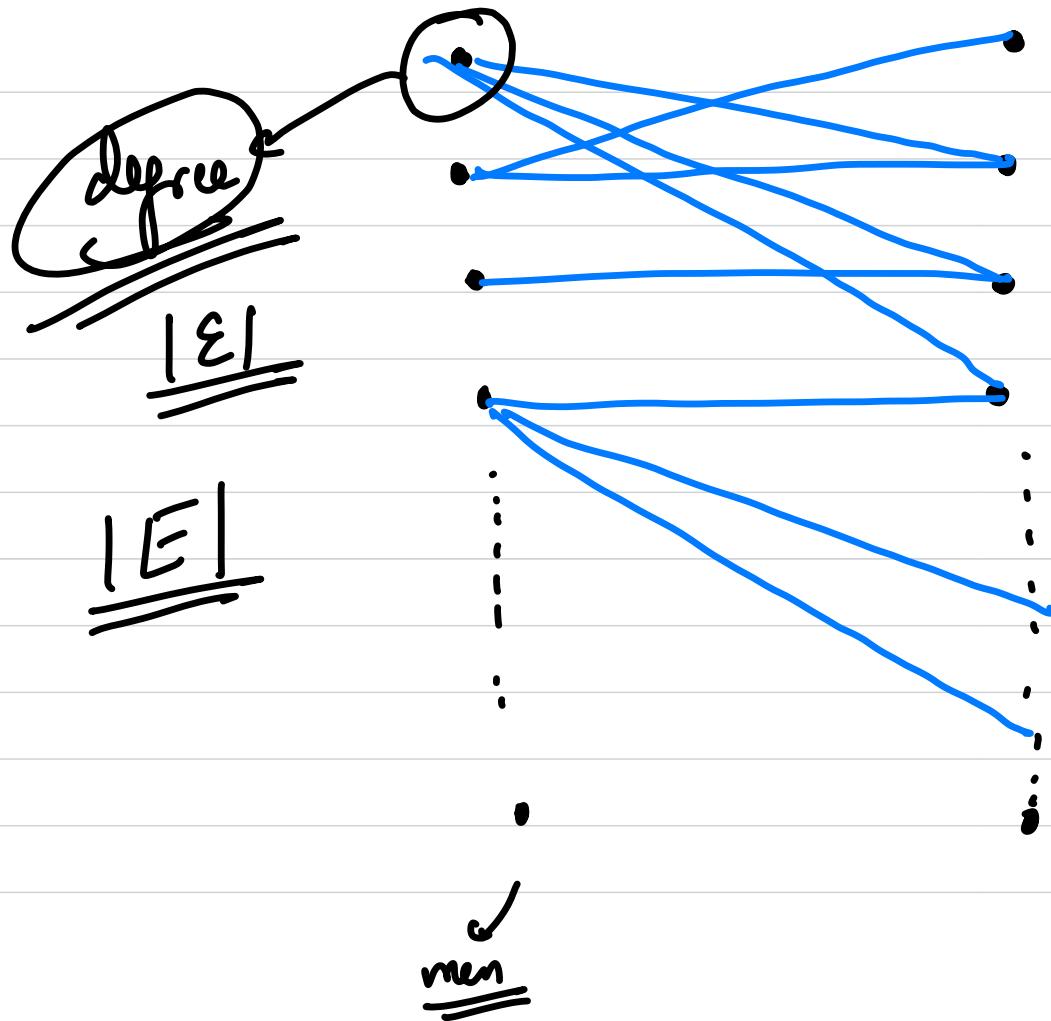


kinematically inf



i.e. male & female count

~~Q~~ Let's say we have a gender dist. n of a country  
i.e. no. of male population & no. of female  
population. Say,  $x$  is a no. that denotes  
average no. of opp gender partners for men  
&  $y$  is denoting average no. of opp gender part-  
ner for women. Then  $\frac{x}{y}$ .



$$x = \frac{\sum_{x \in \text{men}} \deg(x)}{|\text{men}|}$$

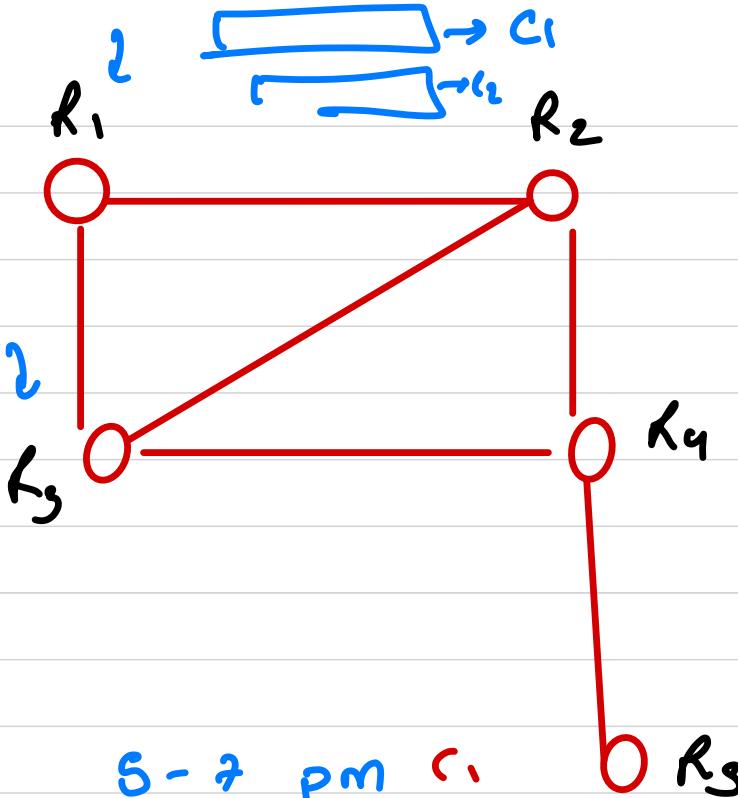
$$y = \frac{\sum_{y \in \text{woma}} \deg(y)}{|\text{woma}|}$$

$$\frac{x}{y} = \frac{187}{170}$$

$$\frac{x}{y} = \frac{187}{170}$$

$$\frac{x}{y} = \frac{187}{170}$$

$\bigcap \sigma$   
meetings



graph colony  
problem

$X(G)$

Range of meeting times

5 - 7 pm	$c_1$
7 - 9 pm	$c_2$
9 - 11 pm	$c_3$
11 - 2 am	$c_4$
3 - 5 am	$c_5$

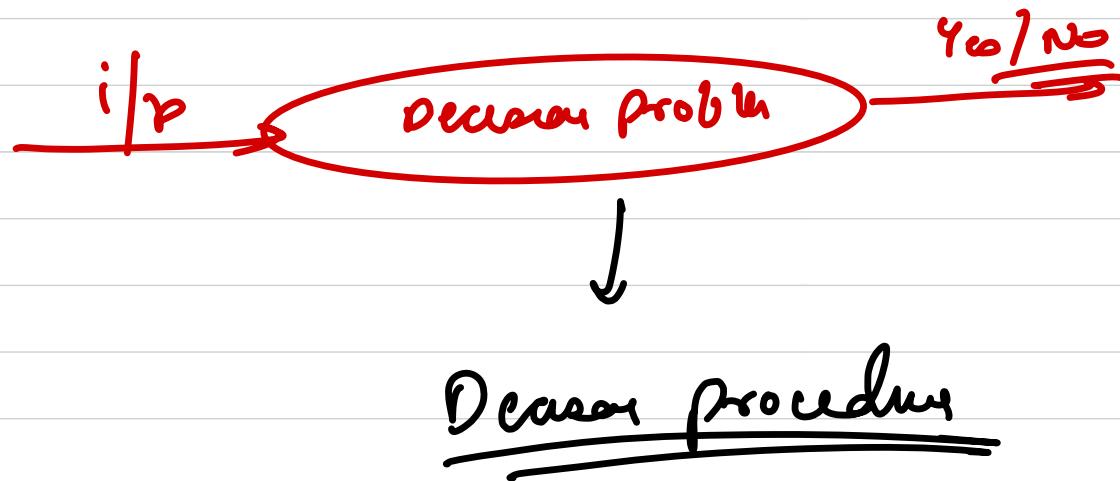
$n$  → no. of nodes

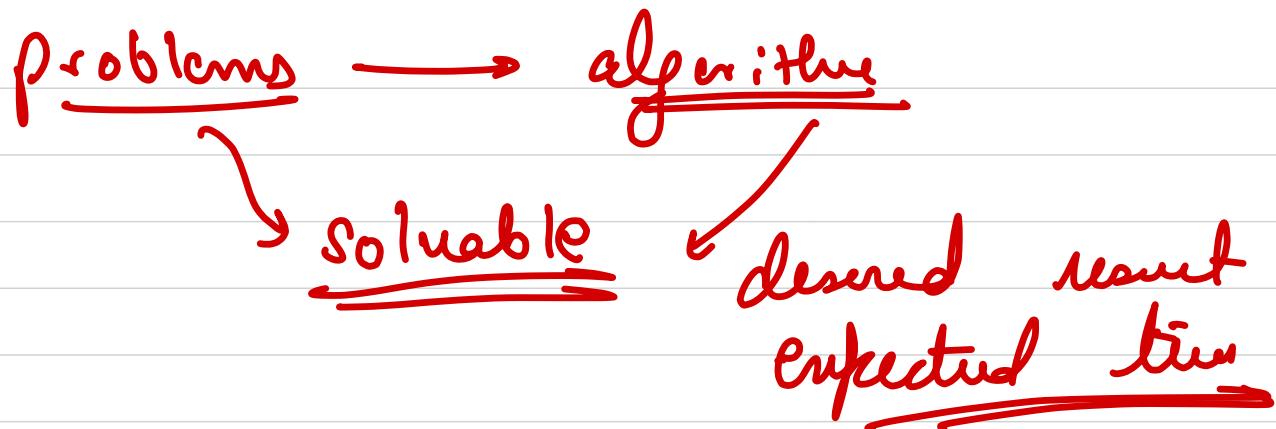
What is a graph colouring problem ??

Given a graph  $G_1$ , and  $k$  colours, assign a colour to each node so that adjacent nodes get different colors.

→ Minimum no. of colors for which the above statement holds true is called as chromatic number of graph.  $\chi(G_1) \rightarrow \min \text{ value of } \underline{k}$

→ Decision problem → yes/no ans associated





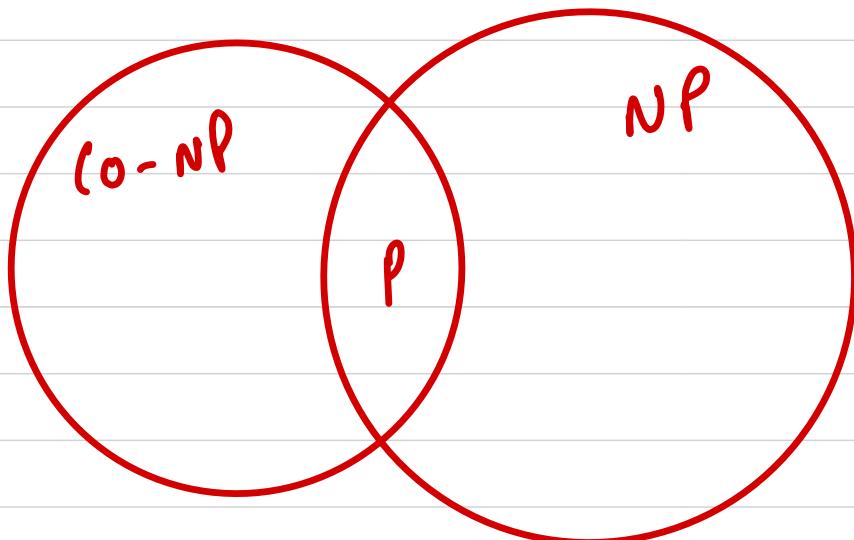
① P-class  $\rightarrow$  set of decision problem which  
are polynomial time solvable -  $\underline{\underline{O(n^k)}}$   
 $\underline{\underline{k \rightarrow \text{const}}}$

NP class  $\rightarrow$  non-polynomial time class  $\rightarrow$   
set of decision problem that are only  
verifiable in polynomial time.

Ex 500 students  $\rightarrow$  100 rooms  
 $\rightarrow$  few students should not  
 $\hookrightarrow$  grouped together.

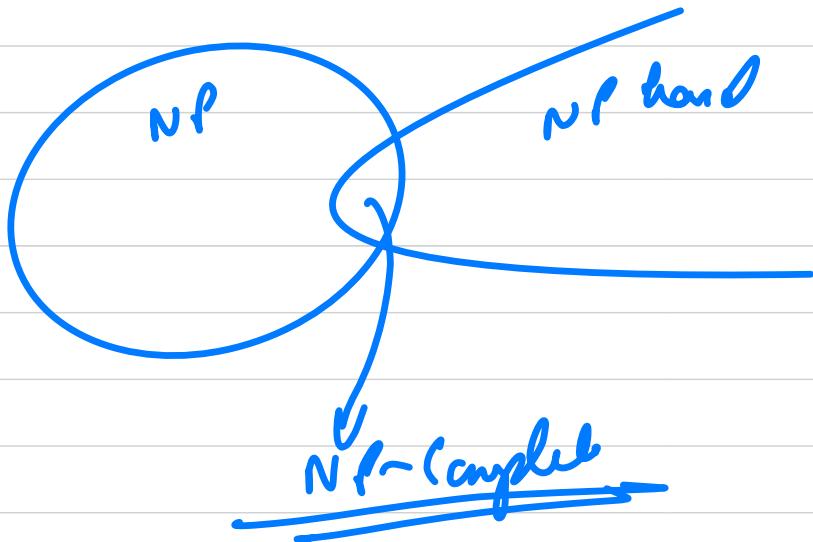
$\text{Co-NP} \rightarrow \underline{\text{"No"}}$

np-hard



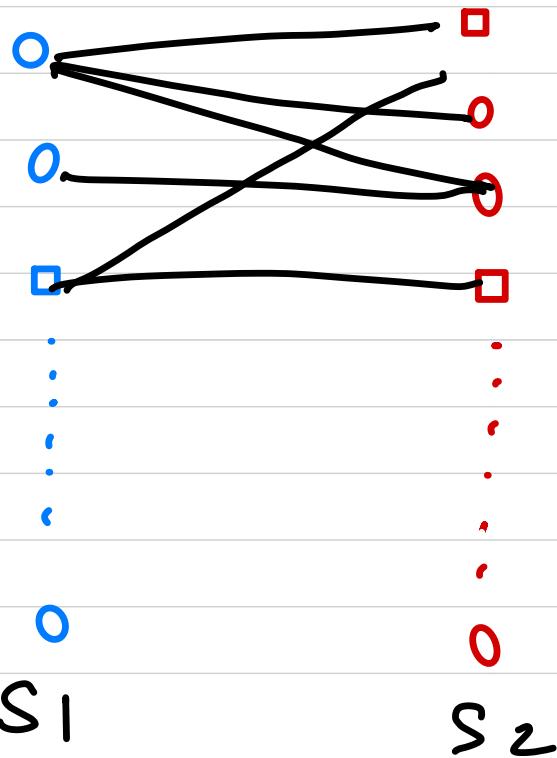
detem  
non-det

NP complete → then lie in both NP & NP-hard



\* Bipartite graph  $\rightarrow$  2 colourable graph

DSU ?



$\rightarrow$  algorithm  $\rightarrow$  colour

with n-nodes

↓

Procedure  $\text{P.C.}$

# Theorem Given a graph  $G$  have a degree  $d$ ,  
then the basic algo uses at most  $d+1$

colors for  $G$ .

upon 6 and

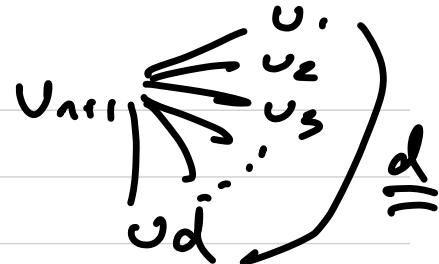
I.H.  $\rightarrow$   $P(n)$   $\leftarrow$  true for  $n$  nodes

$B.C.$   $\rightarrow$   $n=1$

$n$   $\rightarrow$  degree  $\rightarrow$   $0$

$I \rightarrow f(n)$  claims true

Proof  $\rightarrow f(n+1) \rightarrow \underline{\text{true}}$



$G \rightarrow (V, E) \rightarrow (n+1)$  node graph - &

D is the max degree in  $G$ .  
=====

$v_1, v_2, \dots, v_{n-1}, v_n, v_{n+1}$

Remove  $v_{n+1}$  & correspond edges, & make  
a new graph  $G' \leq d$   
To colour  $G'$  all to  $f(n) \rightarrow$  we need  $d+1$  colors  
at most.

$\underline{a^1 \rightarrow b_1} \rightarrow \underline{(d+1)_{10101}}$

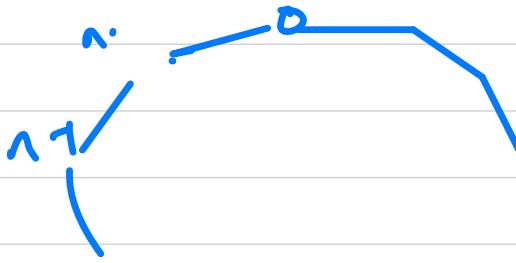
$\underline{\underline{H^L}}$

Given a graph  $\Gamma$  detect if it has odd cycle?:

2 H w

$\Rightarrow$  A bipartite graph has no odd cycle.

Lemma



Proove  $\rightarrow$  A graph  $G$  is bipartite if and only if it contains no odd length cycles.

Part I  $\rightarrow G$  is bipartite  $\Rightarrow G$  has no odd cycles

Ans  $\rightarrow$  Suppose  $G$  is bipartite

$$V(G) = X \cup Y \text{ and } X \cap Y = \emptyset$$

for every edge  $e \in E(G)$   $e = \overset{?}{x} \overset{?}{y}$  when  $x \in X$   
 $y \in Y$

Proof by contradiction:

Assume G has at least one odd cycle.

Let C be a cycle, such that <sup>of odd length</sup>

$$C = \{v_1, v_2, v_3, \dots, v_n\} \text{ when } n \text{ is odd}$$

$v_i \in X$

then  $v_2 \in Y$

$(v_1 - v_2)$

$$\Rightarrow v_i \in \begin{cases} X & \text{if } i \text{ is odd} \\ Y & \text{else} \end{cases}$$

$\Rightarrow$  if  $n$  is odd,  $v_n \in X$ , if  $v_i \in X \& v_n \in X$   
 $v_i, v_n \in E(G) \rightarrow$  This is impossible.

Part 2:  $\rightarrow$  If a graph has no odd cycle  $\Rightarrow$  it is bipartite

$\Rightarrow$  Suppose all cycles in  $G$  is of even length,  
 $\rightarrow$  if graph is disconnected, then we consider  
our proof for any component

Let say  $m \in V(G)$

then let  $X = \{x \mid d_G(m, x) \text{ is even}\}$

( $X, Y$  are 2  
partite set)

$Y = \{y \mid d_G(m, y) \text{ is odd}\}$

since  $G$  is connected  $X \cup Y = V(G)$

$$X \cap Y = \emptyset$$

We need to show any edge  $e \in E(G)$  is of the form  $e = xy \Rightarrow x \in X$  and  $y \in Y$ .

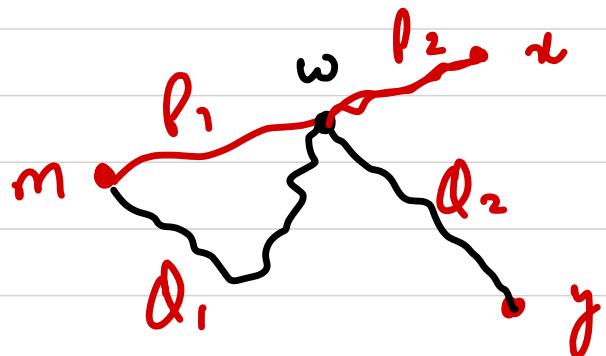
Proof by contradiction:

assume  $\rightarrow$  that  $x, y \in X$  or  
 $x, y \in Y$

assume  $x, y \in X$  and let  $P$ : shortest path  
from  $m$  to  $x$ .

let  $Q$ : shortest path from  $m$  to  $y$ .

Since  $x, y \in X$  both  $P$  and  $Q$  should be even (i.e  
Same parity).



assume  $w$  is the last common vertex of  $P$  and  $Q$ .

$P_1$ : Shortest path from  $m \rightarrow w$

$$P = P_1 P_2$$

$P_2$ : Shortest path from  $w \rightarrow x$

$Q_1$ : Shortest path from  $m \rightarrow w$

$$Q = Q_1 Q_2$$

$Q_2$ : Shortest path from  $w \rightarrow y$

Since  $P$  and  $Q$  are the shortest paths  $\Rightarrow$

$P_1$  and  $Q_1$  are the shortest paths break  
w from  $m$ .

$$|P_1| = |Q_1|$$

Now,  $|P_2|$  and  $|Q_2|$  will have same parity.  
So both  $P_2$  &  $Q_2$  are odd or both are even.  
Thus the path from  $x \xrightarrow{P_2} w \xrightarrow{Q_2} y$  is always even  
parity.

Now, if  $x, y \in E(G)$  then  $P_2 \sim Q_2$  together  
with  $xy$  will form a cycle of odd lengths.  
This is contradiction as  $x, y \notin E(G)$

Q2 Given a matrix of  $M \times N$  binary values. find if we can have a 1D array B such that

$$M[i][j] = [B_i - B_j] \geq 0$$

	1	2
1	0	1
2	1	0

3	4
---	---

Can we treat 0's & 1's differently

Binary Matrix



given matrix  $\rightarrow \underline{\underline{M}}$

$$M_{ij} = |B_i - B_j|$$

is there an array  $B$ , such that

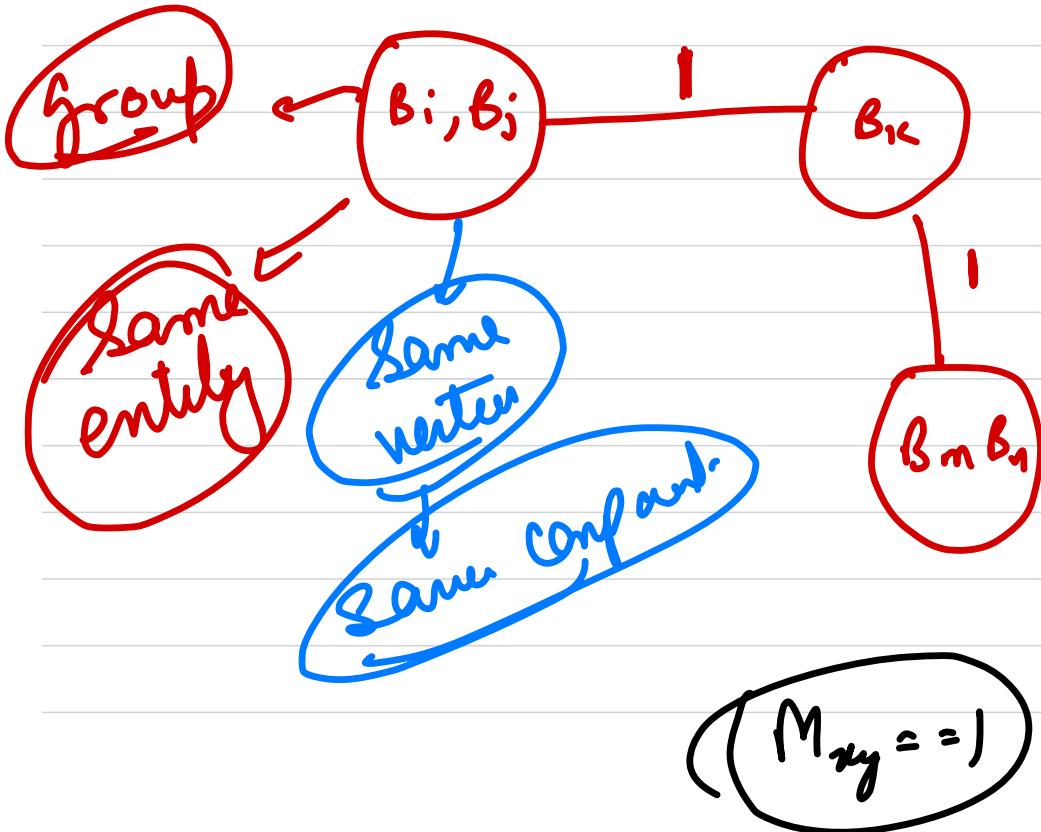
$$M_{ij} = M_{ji}$$

(observe)

$$|B_i - B_j| = |B_j - B_i|$$

$M_{ii} = 0$   
 $\hookrightarrow \text{false} \rightarrow \text{false}$

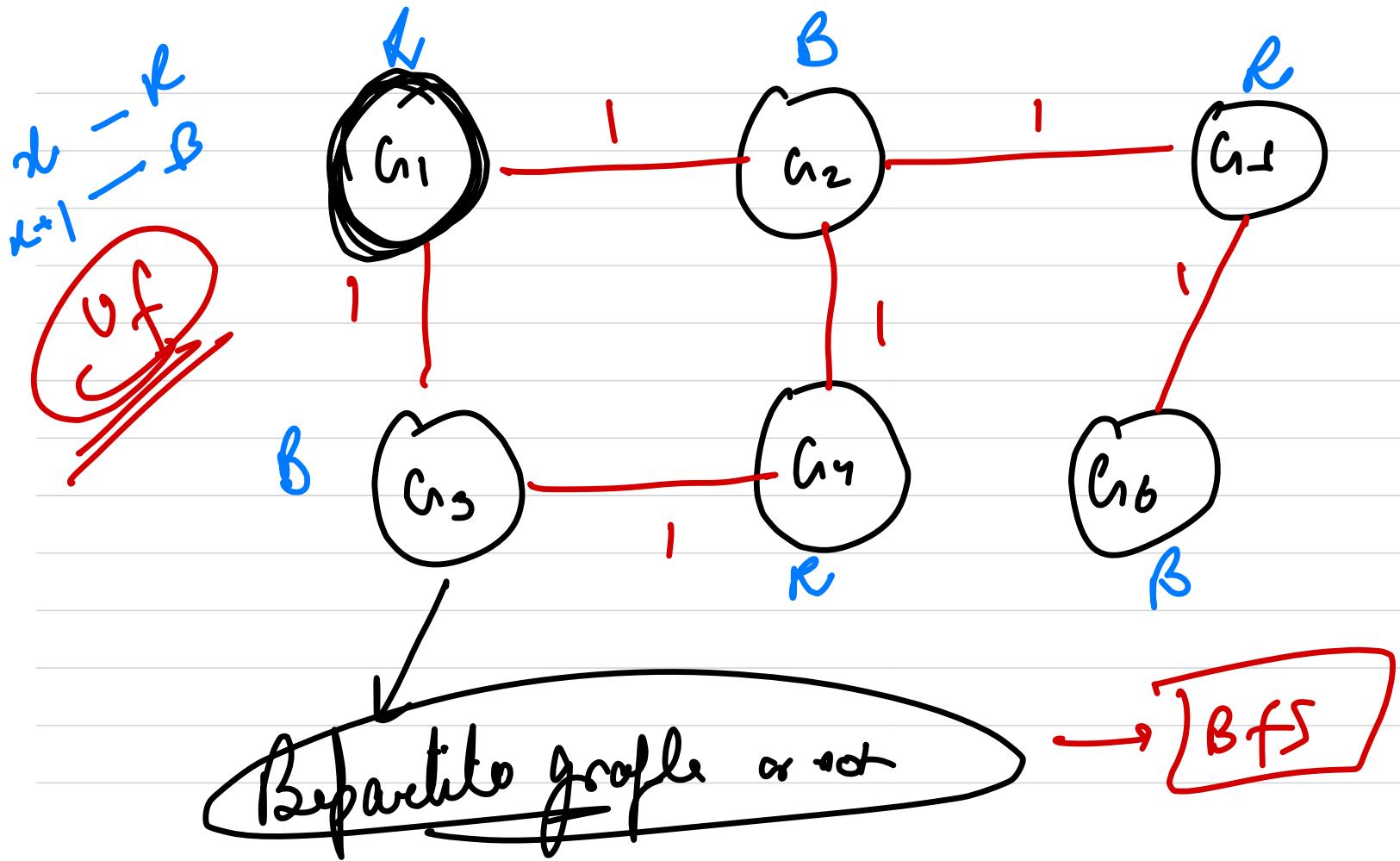
we know if  $M_{ij} = 0$  then  $B_i = B_j$



$$M_{ik} = 1$$

$$B_i = B_k$$

$$M_{ij} = 0$$

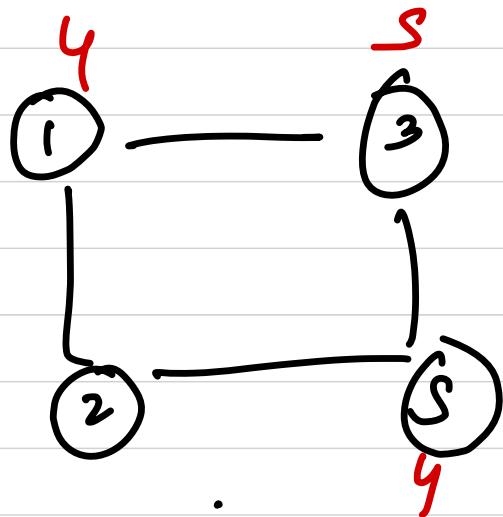
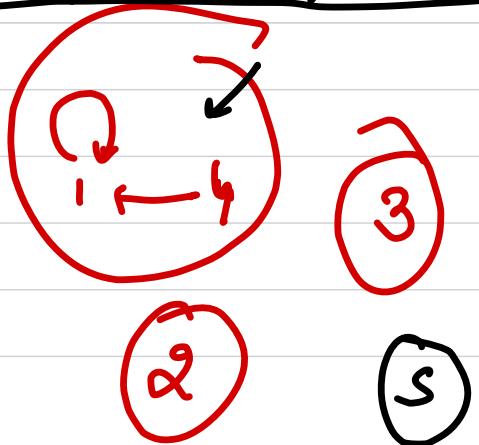


→ Using DSU, merge all the vertex in the corresponding groups

→ Every group has a leader, so use the leaders make a graph

→ Check if it is Bipartite

	1	2	3	4	5
1	0	1	1	0	1
2	1	0		1	1
3	1	0	1	1	1
4	1	1	1	0	1
5	1	1	1	1	0



$\Rightarrow$



1

$1 \rightarrow \text{ev}$

1

$2 \rightarrow \text{od} \checkmark$

1

$3 \rightarrow \text{ev}$

2

$1 \rightarrow \text{od} \checkmark$

2

$2 \rightarrow \text{ev}$

2

$3 \rightarrow \text{od} \checkmark$

3

$1 \rightarrow \text{ev}$

3

$2 \rightarrow \text{od} \checkmark$



2

3

3

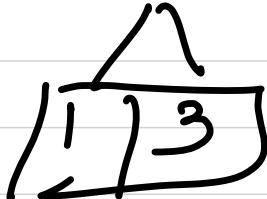
2

1, 2

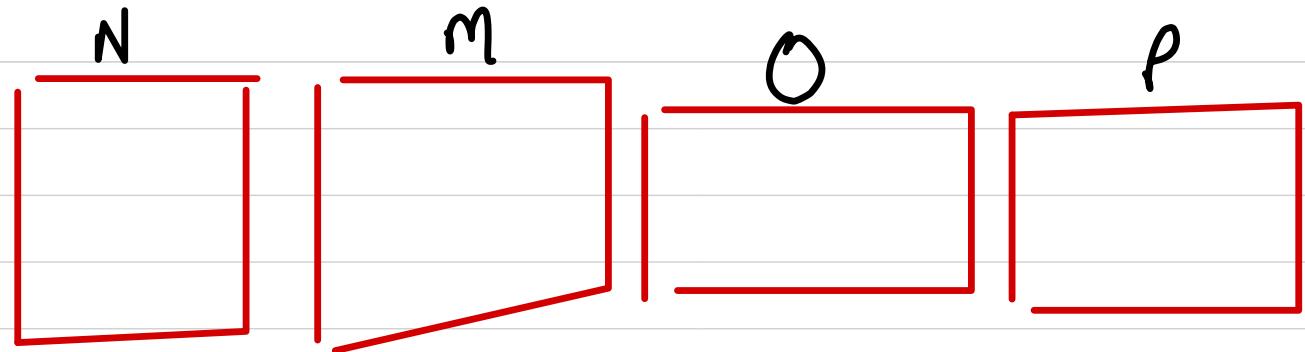
3, 2

$3 \rightarrow \text{ev}$

$\Rightarrow \text{odd} + \text{even} \rightarrow \underline{\text{odd}}$



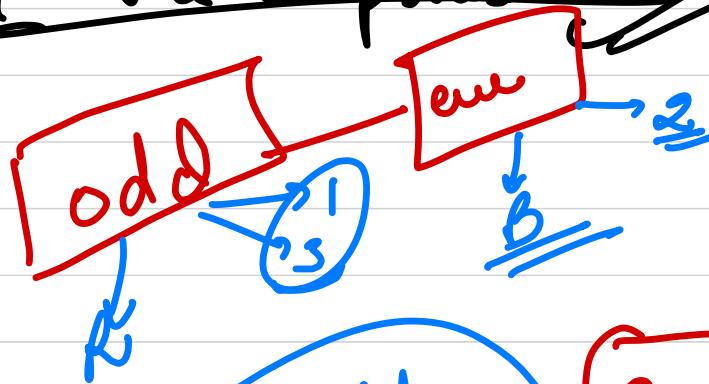
$\Rightarrow$  ~~disconnected graph~~



Product of  $N, M, O, P$

How to solve for  
one component

Pantite  
site

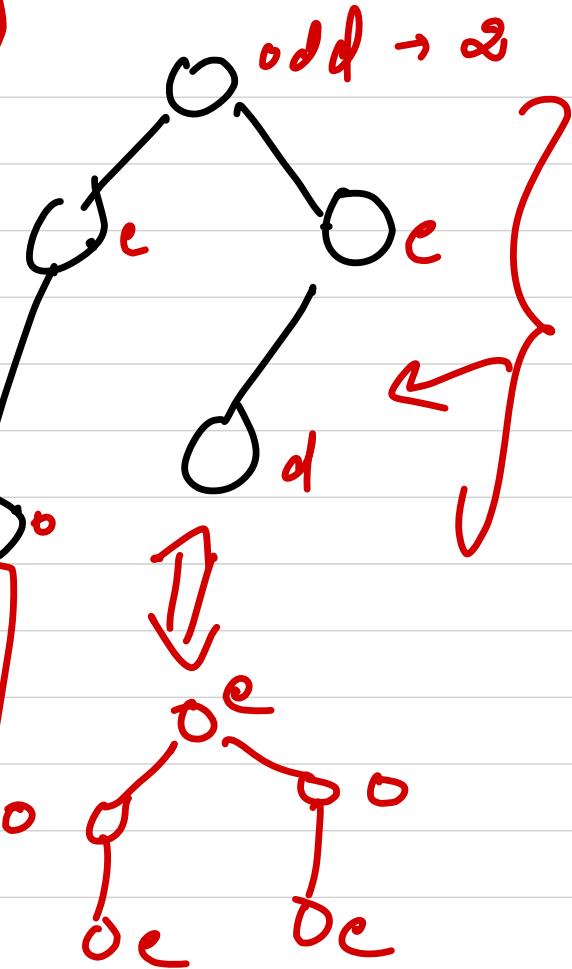


count-R

Bipartite  
graphs

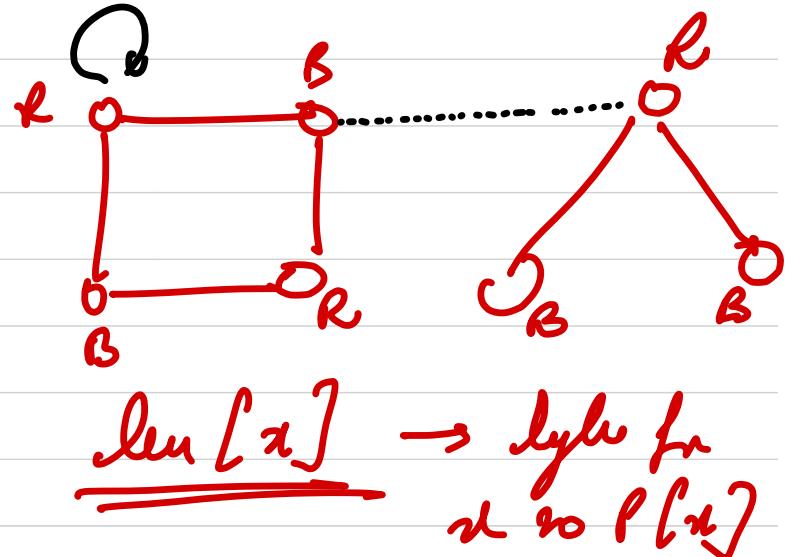
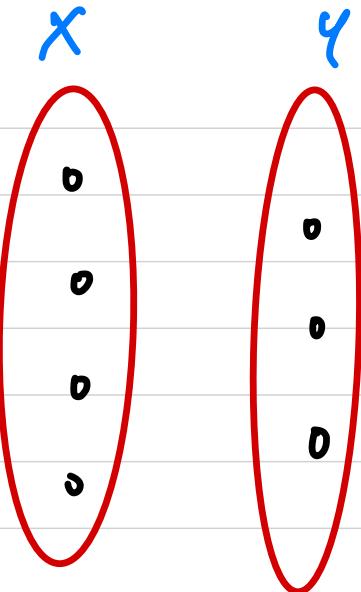
$(2 \times 1 \times 1 \times 2 \times 2)$   
+  $((2 \times 2 \times 1) \times 1)$

count-B



$$\text{Ans}_c \rightarrow 2^{\text{cont\_L}} + 2^{\text{cont\_R}}$$

legen

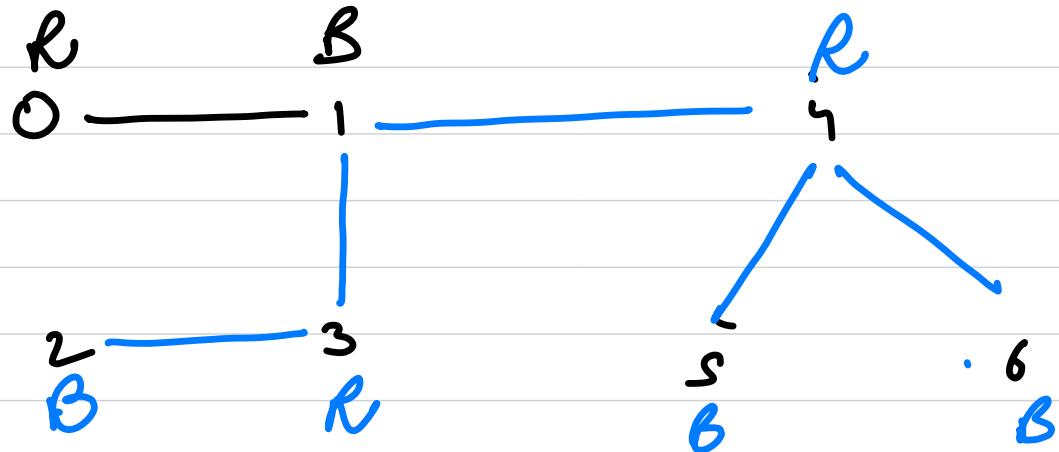


$\rightsquigarrow$

$m, n \in X$   
 $mn \notin E(G)$

$m, n \in Y$   
 $mn \notin E(G)$

~~graph~~



R → even

B → odd

check B-pair  $\rightarrow \underline{\underline{O(j \cdot \log j)}}$

## Dp on graphs

~~DAG~~ DAG CNT2 - Spoj → Directed acyclic graph.

$\sum_{\text{# nodes}}$  weights (dist)

All dist reachable through nodes included

Res node.



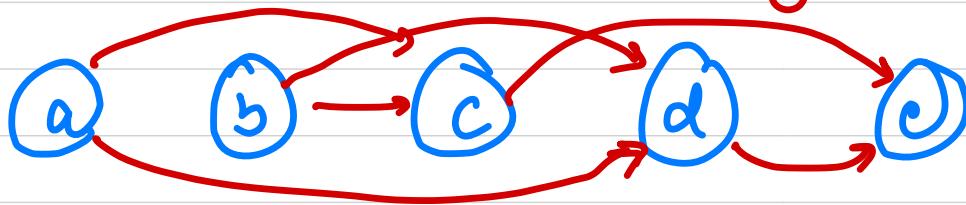
for dp on graphs (dag) we can use topological

sort.

why topological sort??

any dag can be resolved in topological order

which shows dependency graph.



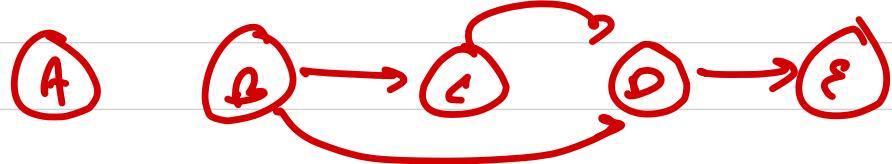
all edges  
will go forward

topological order

for DP (consider bottom up (TD) ) order matter

The order of result of subproblems  
reqd.

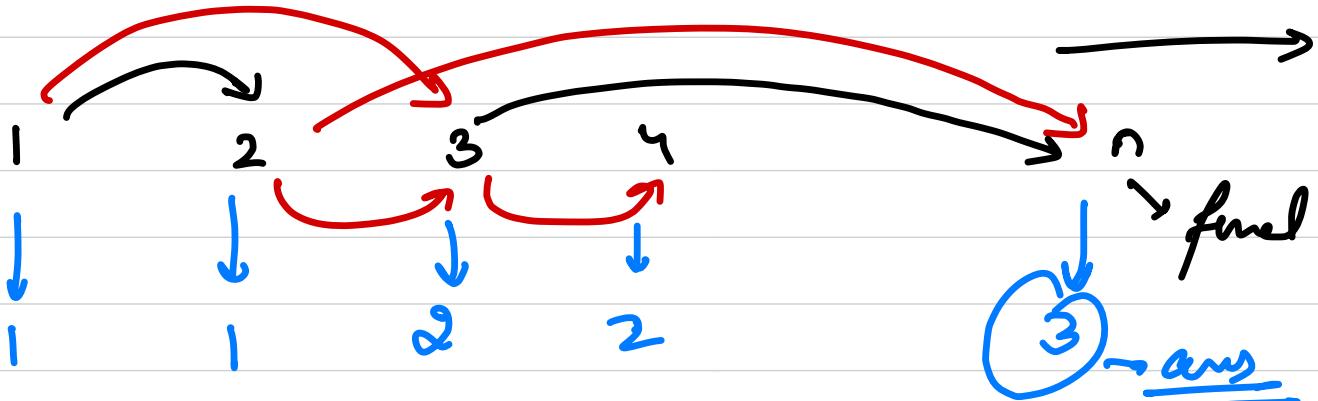
for day we can get dp order using topological  
sort + ..



Topo sort

~~Dog~~

initial



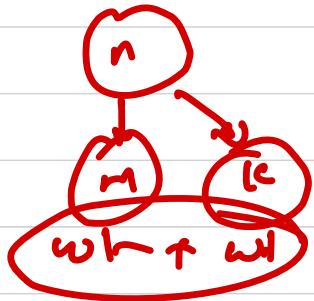
$f(i)$

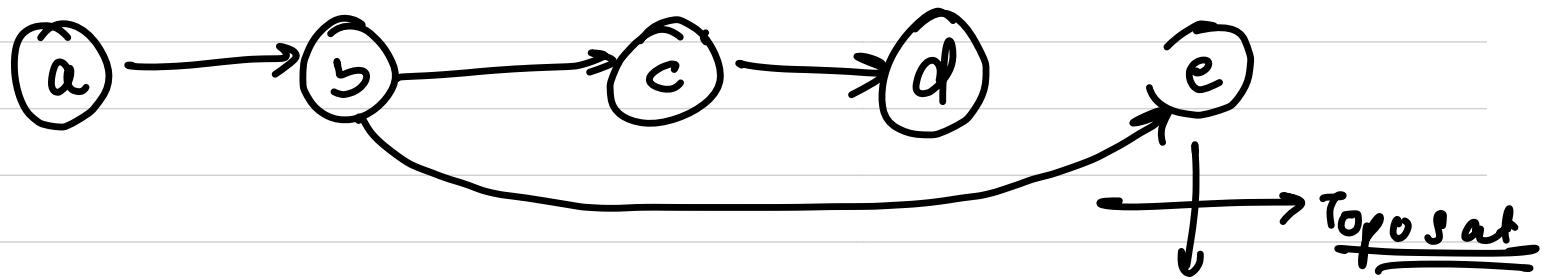
$i \rightarrow j$  ways

~~deg  $c^{n+2}$~~

let's calculate for every node

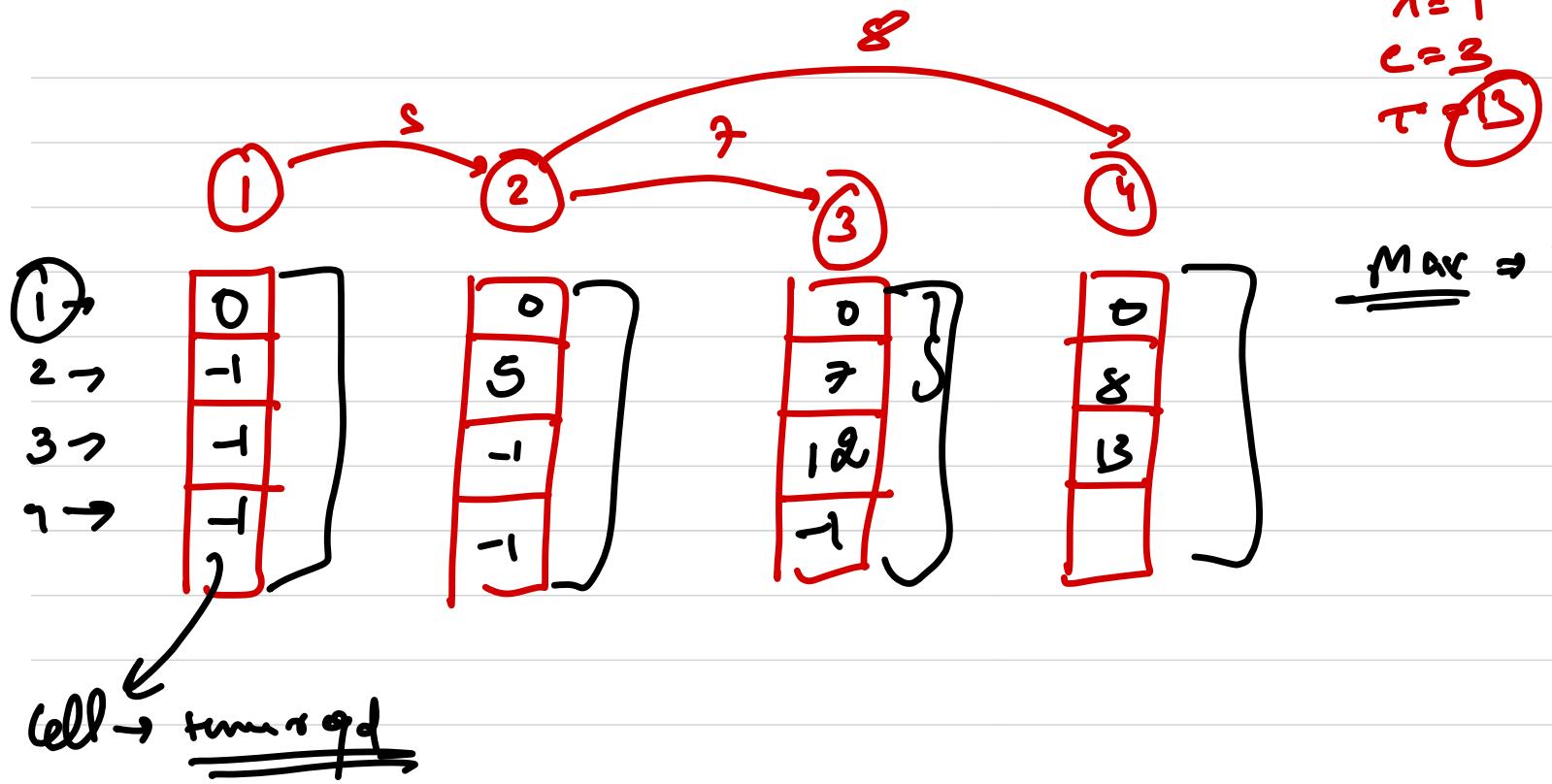
how many other nodes are approachable





1	0	0	0	0	a
1	1	0	0	0	b
1	1	1	0	0	c
1	1	1	1	0	d
1	1	0	0	1	e

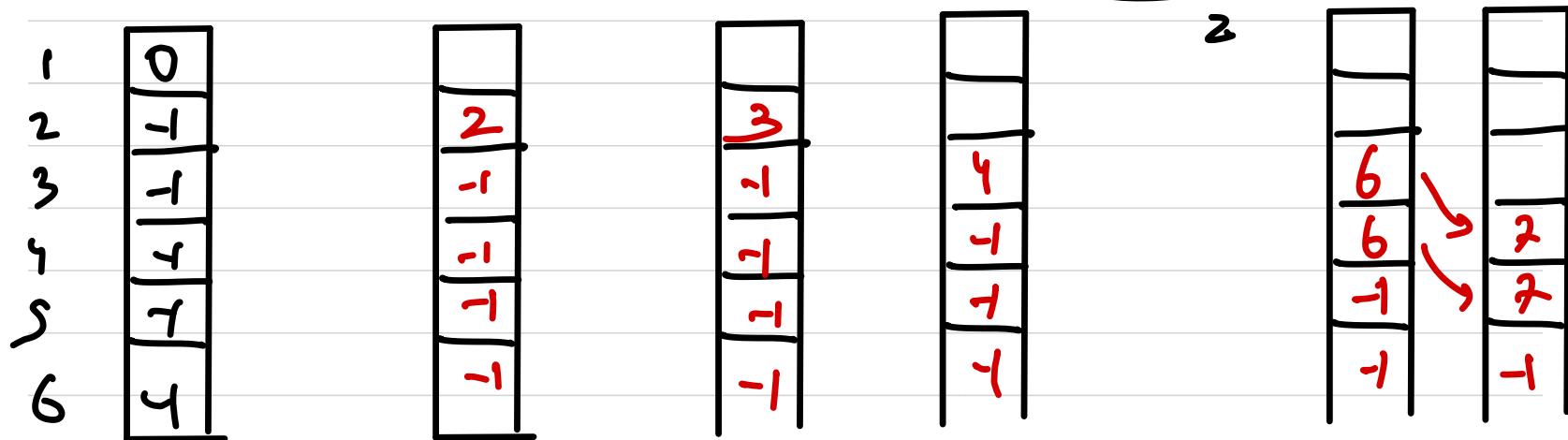
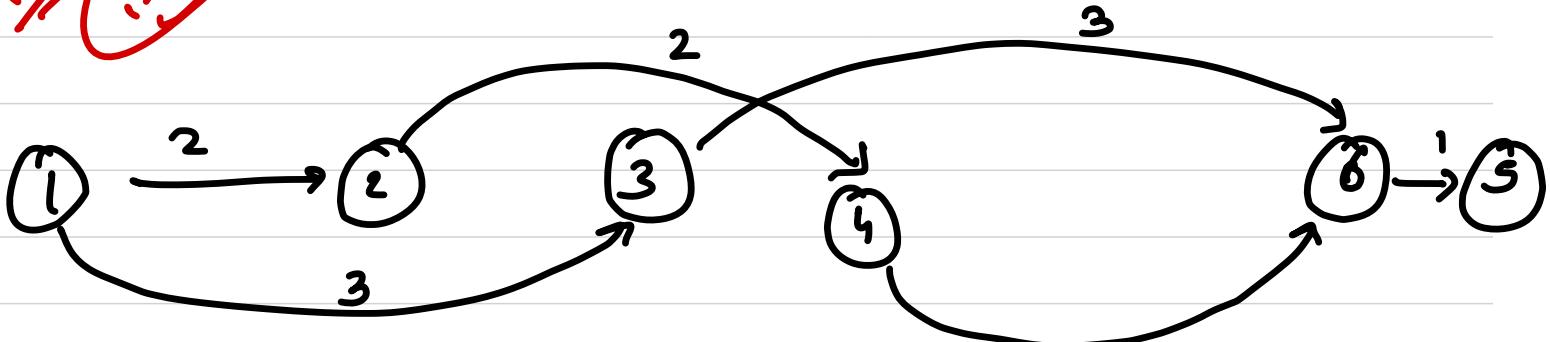
$4 \times 10^8$  → optimale ??



for any node seen in my edges

$\text{N} \rightarrow \text{G}$

$T = 2$



Q. 2 Given a matrix of  $n \times m$  cells & dimensions  
Each cell is either 1 or 0. Two cells are  
connected if they share an edge. The matrix is  
called connected if all 1 cells can be traversed  
without using a 0 cell. Find min no. of 1  
cells that must be replaced by 0 to make  
matrix disconnected.

$$\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array} \rightarrow 2$$

	0	1	2	3
0	0	1	1	0
1	1	0	0	1
2	0	0	0	1
3	0	1	1	0

Max arms will  
~~always be  $2^k$~~

$2^{k-2}$

$\rightarrow 0 \rightarrow$  if # of concavities  $> 1$   
 $\rightarrow 1 \rightarrow$  if there is an antipodal point

$\hookrightarrow \underline{2}$

