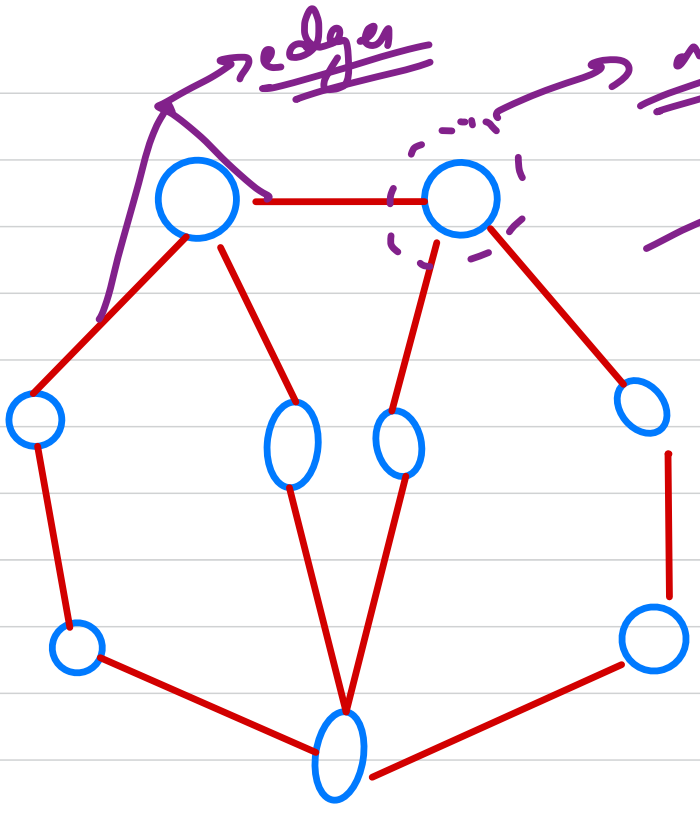
Graphs

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.

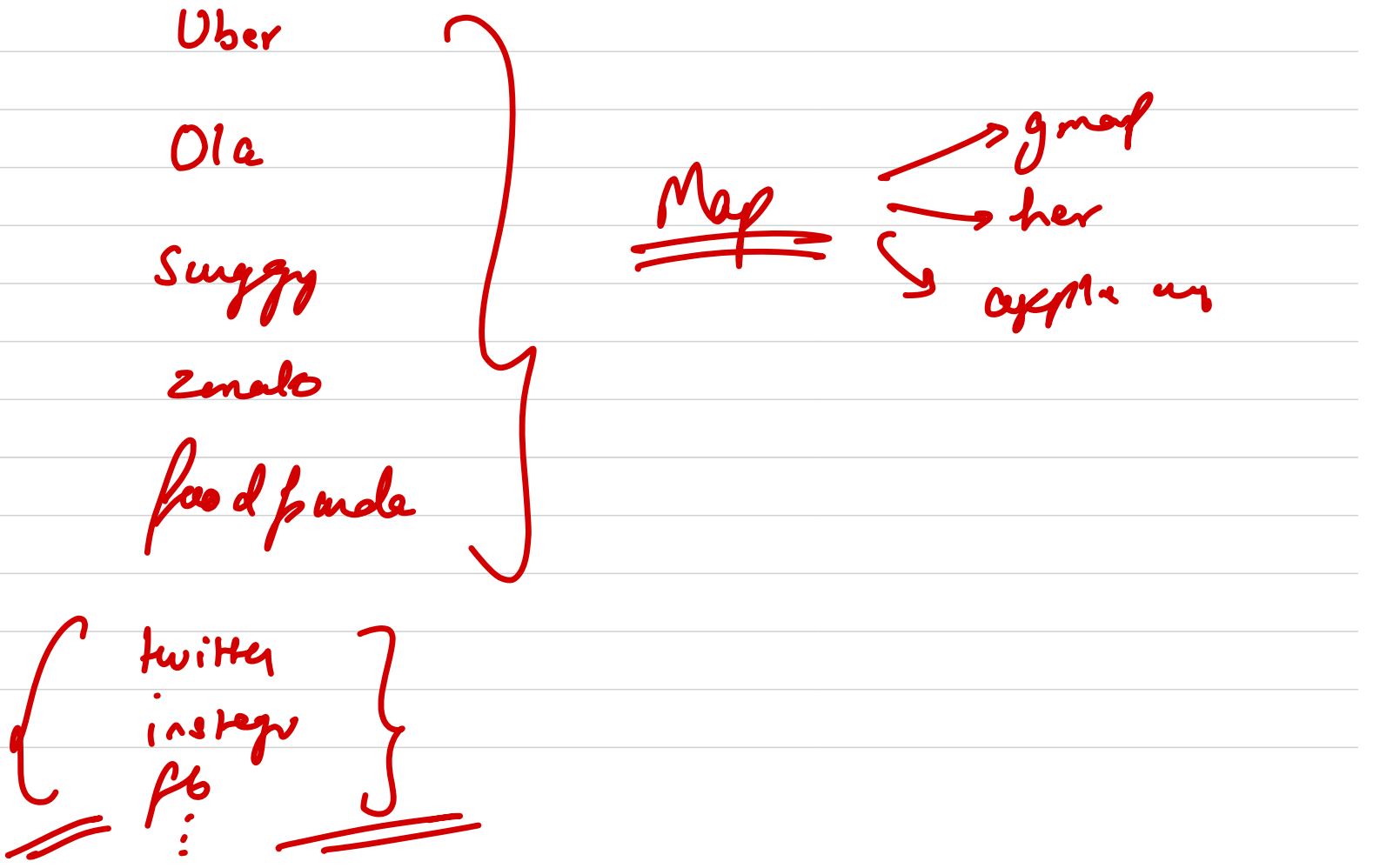


A typical graph

Biology → metabolic networks
Biology → PPI (Protein Protein Interaction)

Electrical → Circuit Organization

CSE → shortest path
CSE → Routing algo
CSE → graph dbs

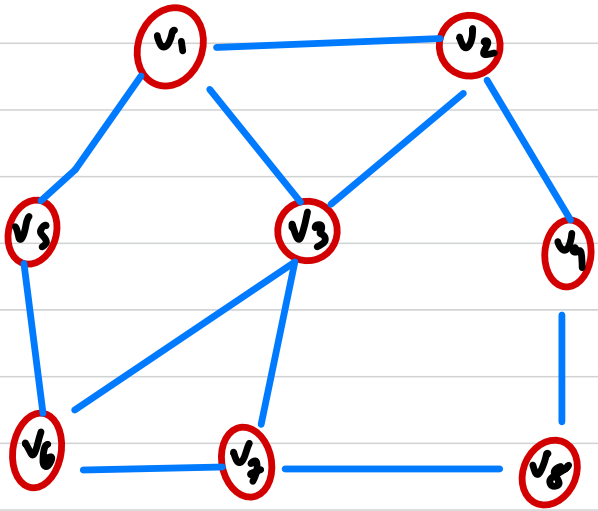

Uber
Ola
Swiggy
Zomato
food panda
Map
gmap
here
twitter
insta
fb

formal definition of graph :

$V = \{v_1, v_2, v_3, v_4, \ldots, v_8\}$

$E = \{\{v_1, v_2\}$
$\{v_2, v_4\}$
$\{v_2, v_3\}$
$\vdots$
$\}$

edges pairs unordered



8 vertices
11 edges

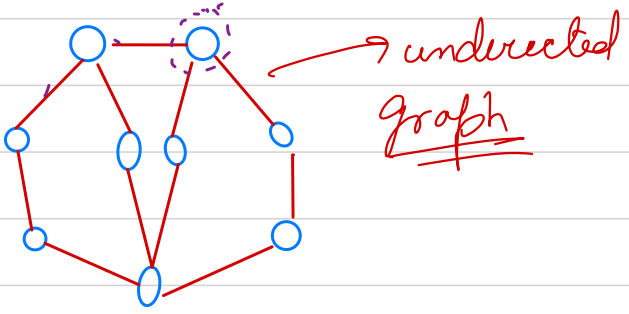
$\rightarrow G = (V, E) \rightarrow$

mathematical notation of a graph

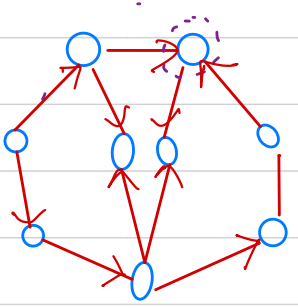
A graph G is an ordered pair of a set V vertices & E, a set of edges.

Types of graph :

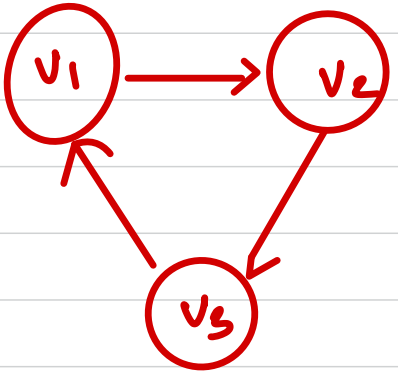
1. Undirected → Facebook
2. Directed → Instagram, Twitter



→ undirected graph



→ Directed graph



$G = (V, E)$

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

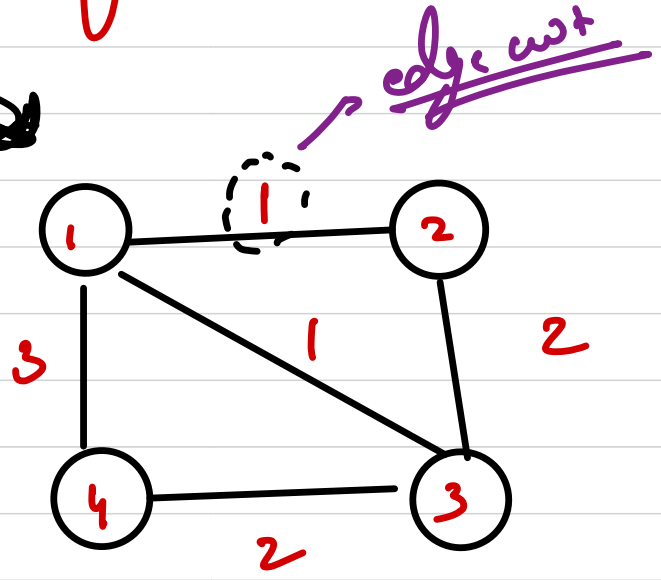
$\{v_2, v_3\}$

$\{v_3, v_1\} \}$

set of ordered pairs

Based on edge property

1. Weighted
2. Unweighted



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

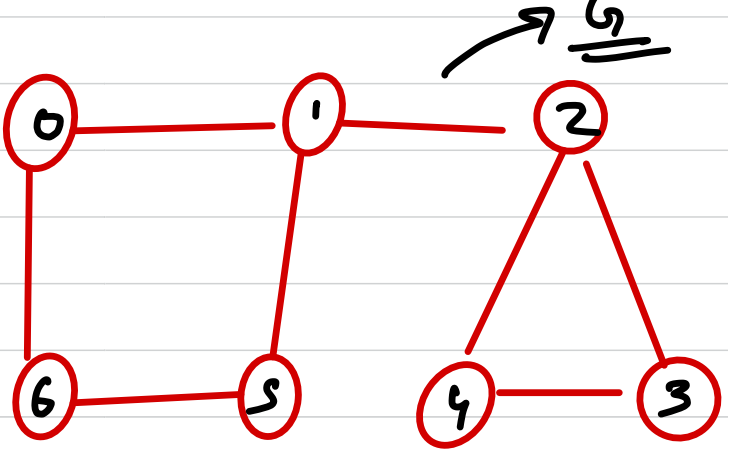
(2) Adjacency List ⟶ Adjacency Map

(3) Incidence Matrix

↳ edge List

Adjacency Matrix

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



A =

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

$V \times V$

→ $O(V^2)$

$(V \times V) \rightarrow$ dimension

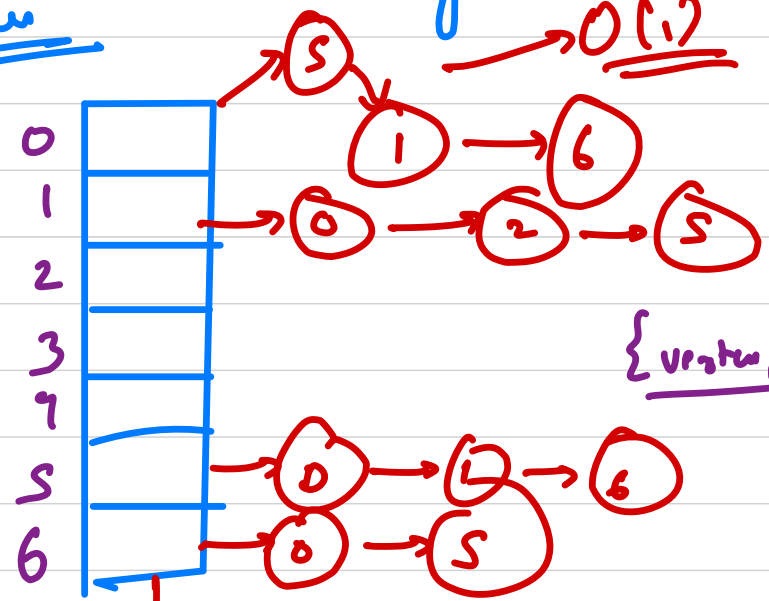
$|V| \Rightarrow$ no of vertices

Adjacency list

Array of LL → space optim

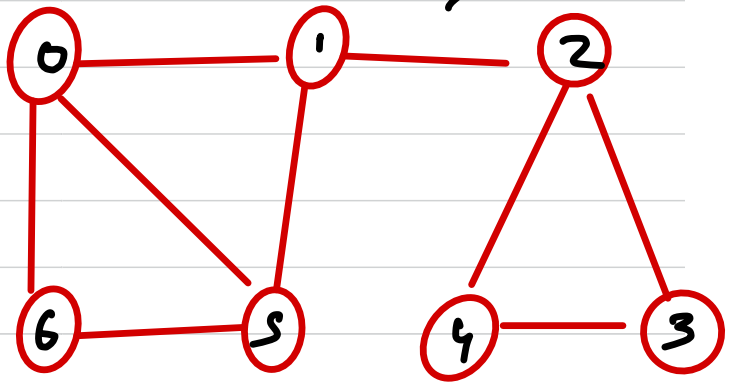
Array → vertex

LL → edges



add at head

good for sparse G



{vertex, wt} → node

$(V + 2E) \approx O(V+E)$

→ ith index store info about ith edge

adjacency map

↓

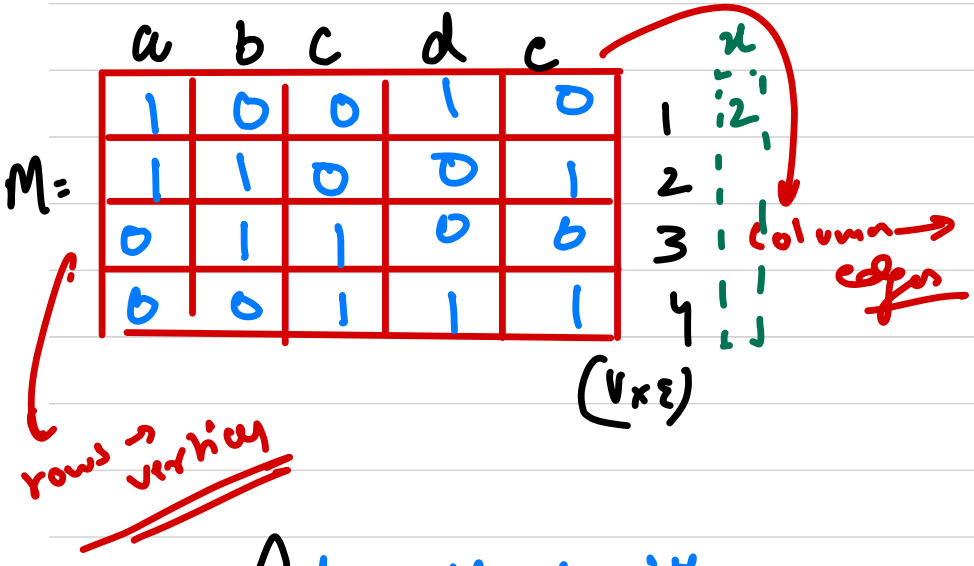
array of hashmap

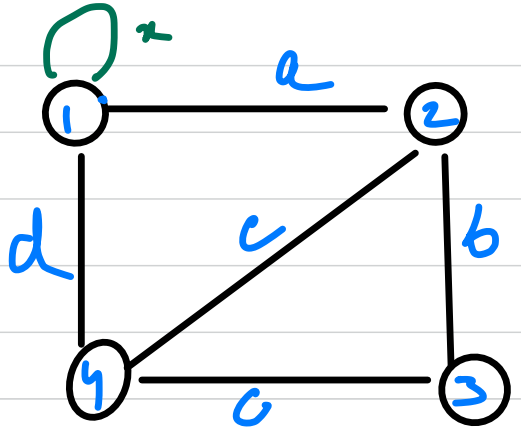
hasedge (u,v) → O(1)

Incidence Matrix

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

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





$(V \times E)$ dimension

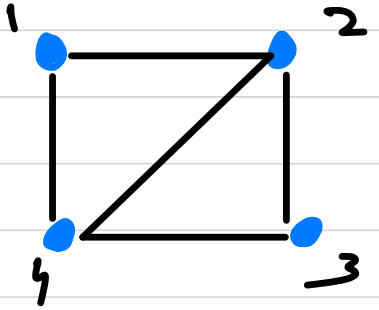
$$M_{ij} = \begin{cases} 1 & \text{if the } i^{th} \text{ vertex belongs to the } j^{th} \text{ edge.} \\ 0 & \text{else} \end{cases}$$

rows_sum = degree

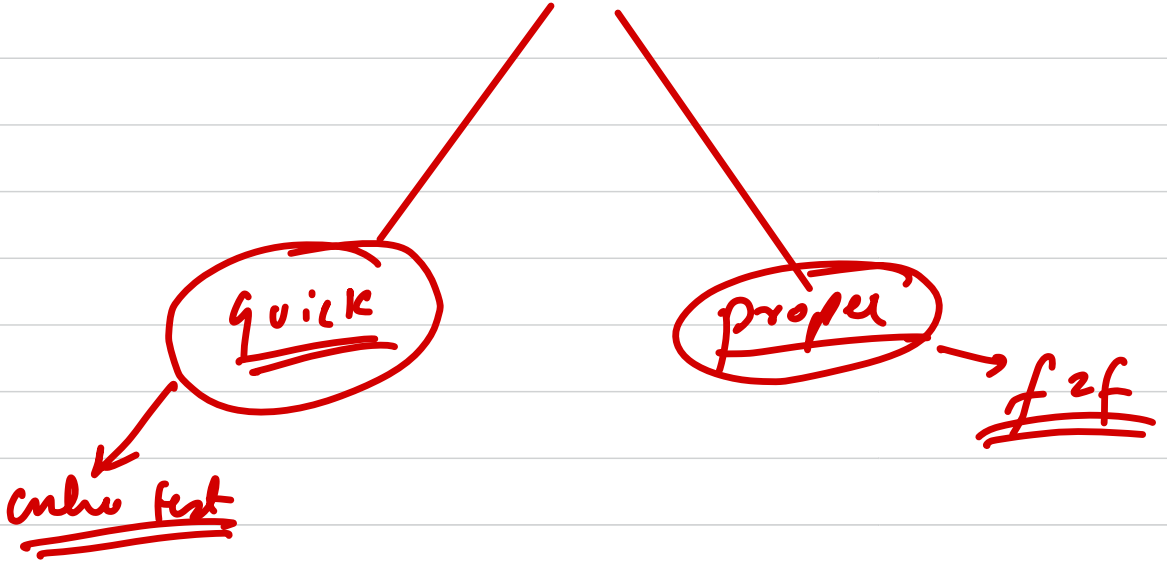
column_sum = 2

Q → What is a degree ??

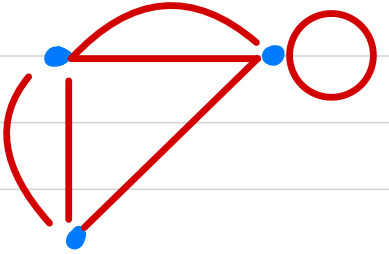
Degree of a vertex in a graph G, is the total no. of edges incident to it / associated with it (undirected graphs)



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

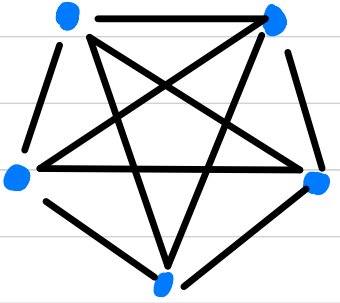

Quick
proper
f2f
online test

Multi graph → an undirected graph with multiple edges and loops allowed.



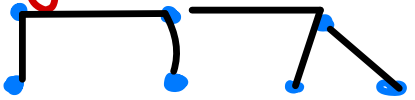
Simple Graph → An undirected graph in which both multiple edge & loops are not allowed.

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

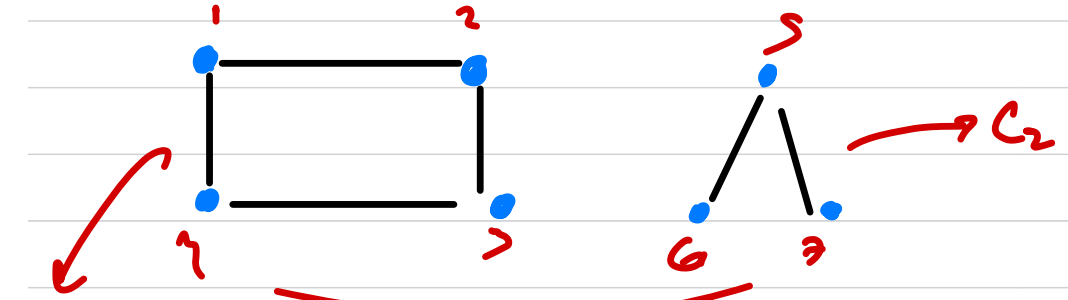


↙ complete graph

Connected graph → A connected graph has a path from every vertex to other vertex, not necessarily direct.



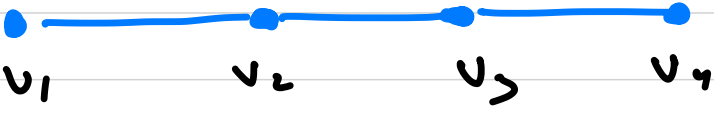
Disconnected graph → at least 2 vertices not have a path to every other vertex.



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

component → a subset of a disconnected / connected graph which is connected.

path → A path P_n is a graph whose vertices can be arranged in some sequence



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

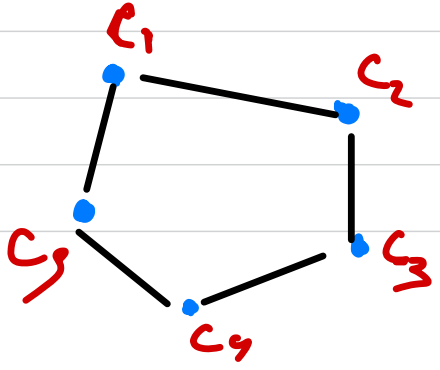
such that edge set of a graph is

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

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

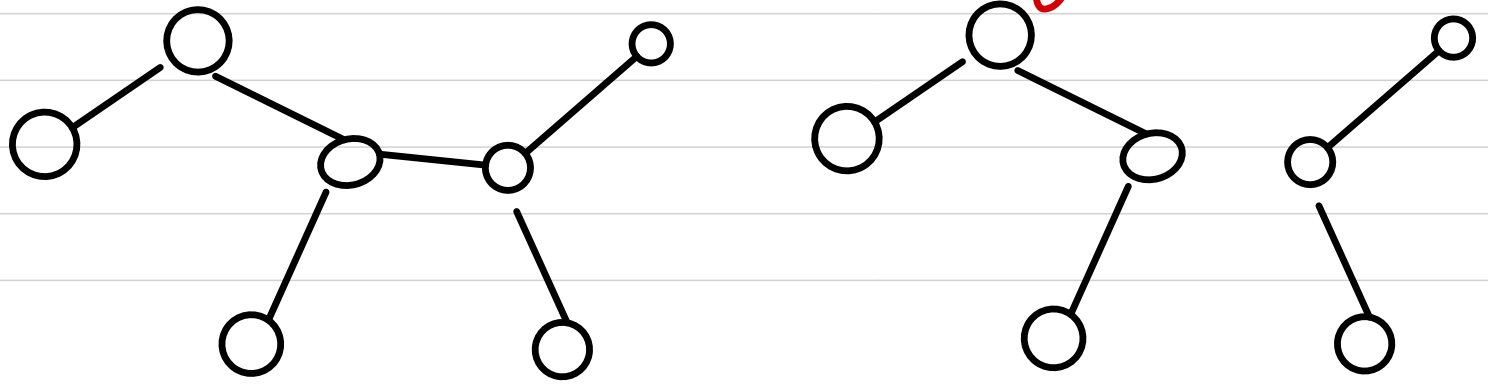
$V = \{v_1\ v_2\ v_3\ v_4\ \ldots\ v_n\}$ Such that edge set is

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

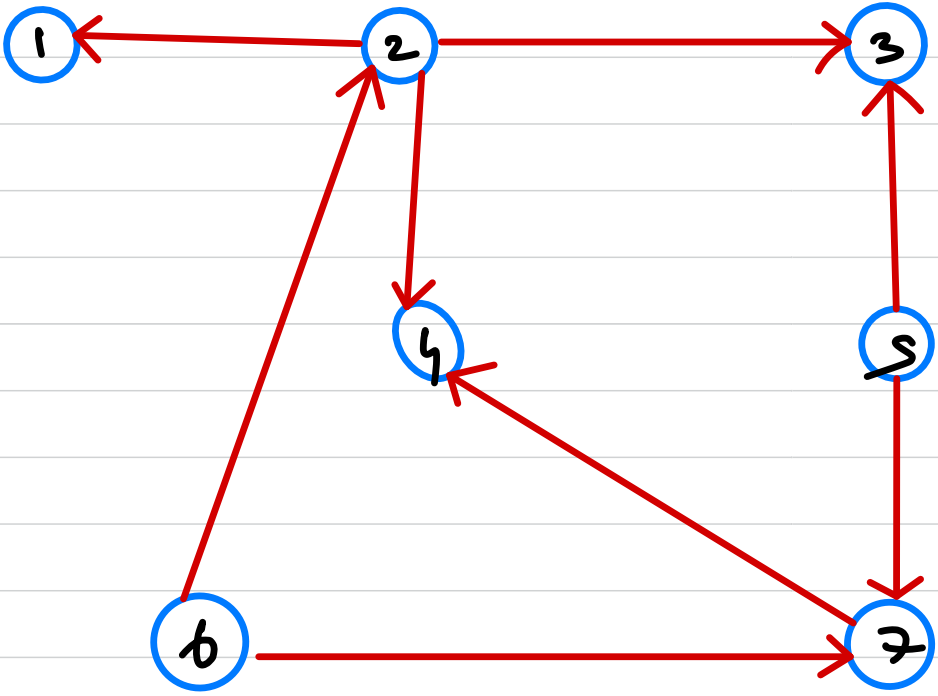


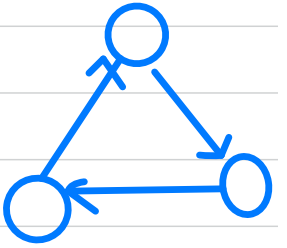
TREE → Tree is a connected graph with no cycles.

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



DAG (Directed acyclic graph)





not a dag

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

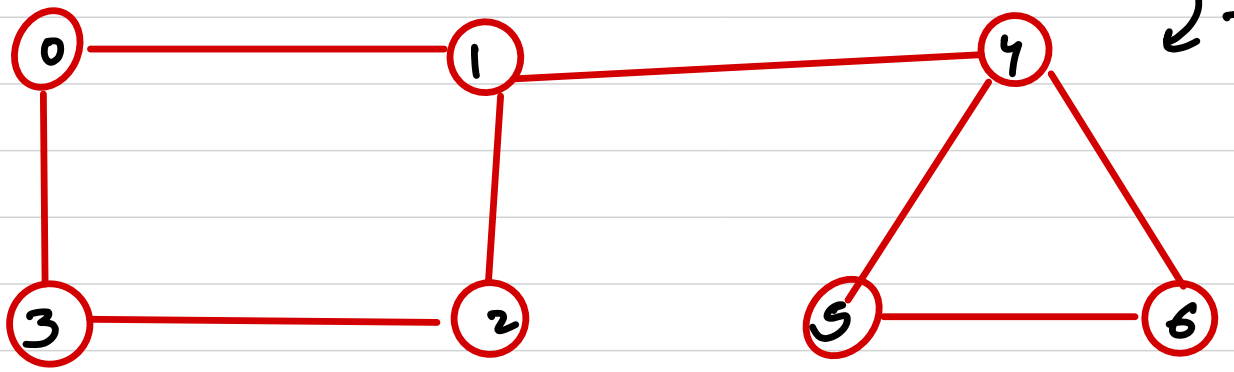
Depth first Search

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.



G

is there a path from 0 to 5 ??

What is the most correct query for this ??

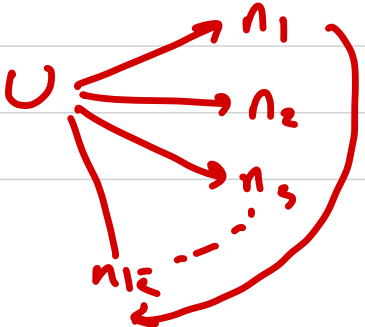
→ neighbours

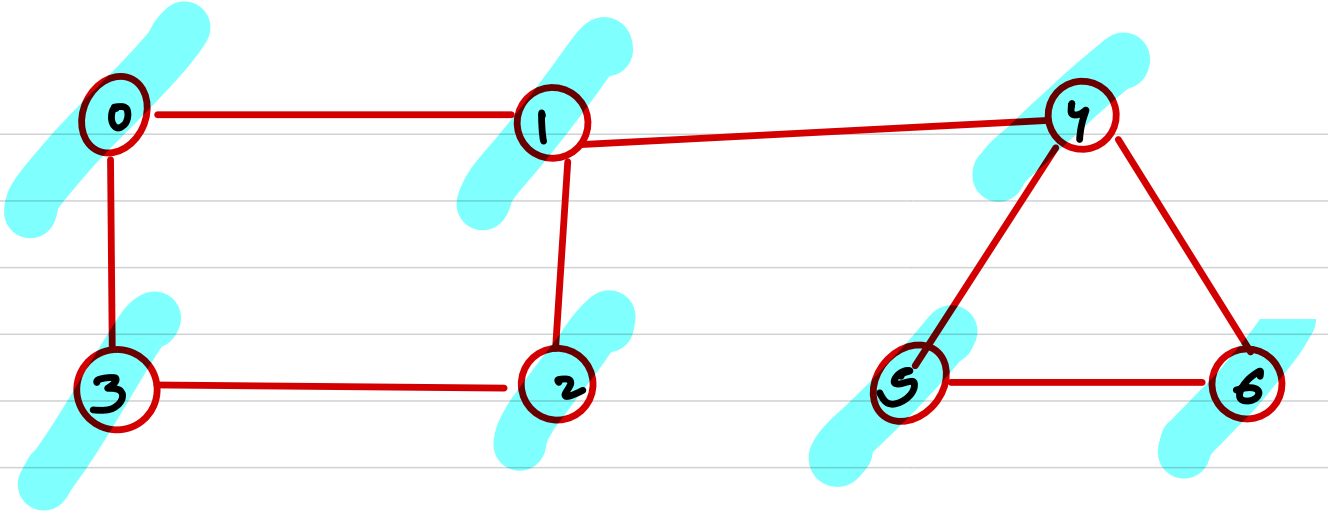
path always exist for immediate neighbour

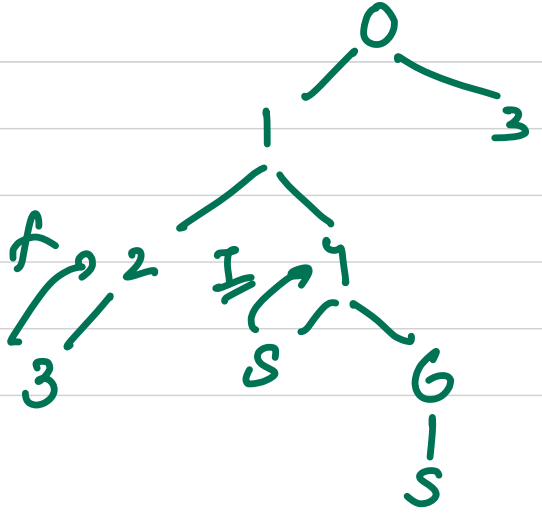
$f(u, v)$ = ?

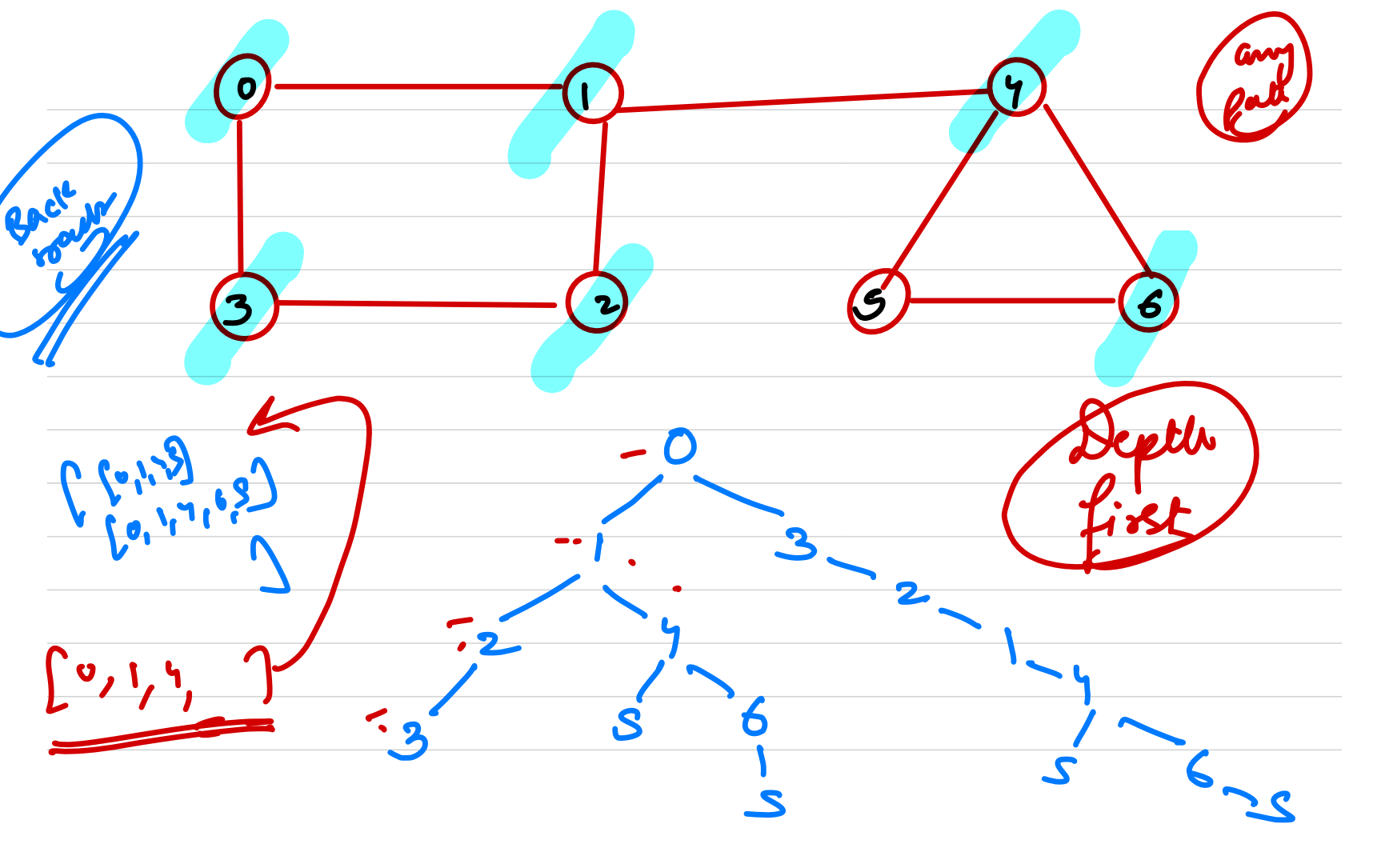
whether there is a path from u to v.

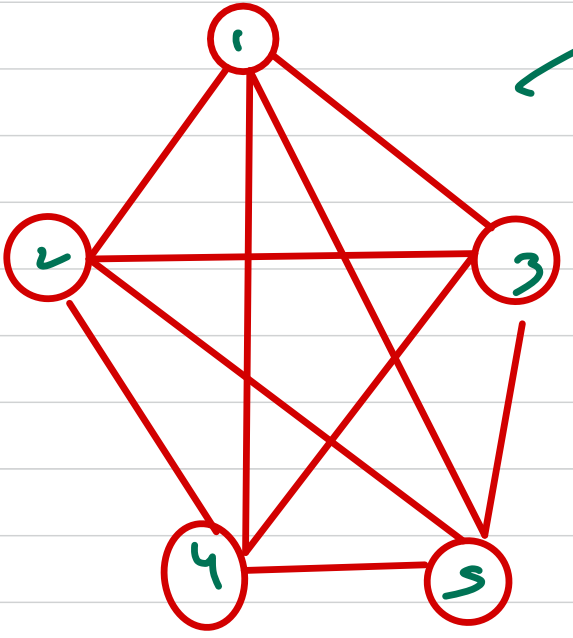
$$f(u,v) = \begin{cases} f(n_1, v) \\ \text{or} \\ f(n_2, v) \\ \text{or} \\ f(n_3, v) \\ \vdots \\ f(n_k, v) \end{cases}$$




0
1
4
3
2
5
6


0
1
3
2
4
3
5
6
5


0
1
4
3
2
5
6
Back track
any path
Depth first



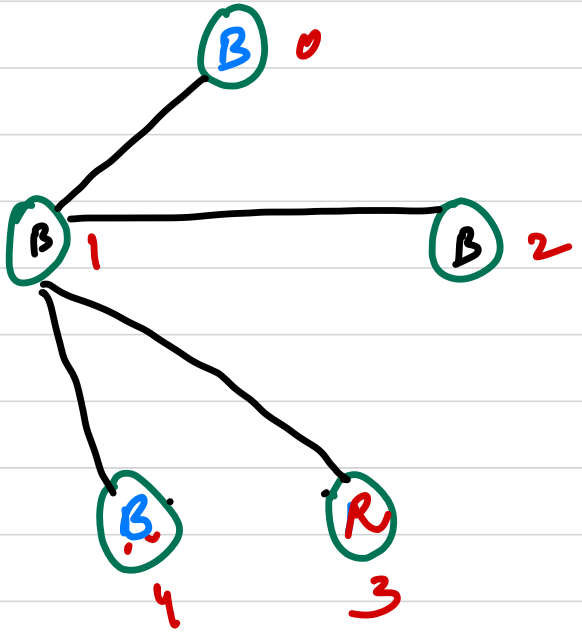
→ complete graph

1, 5

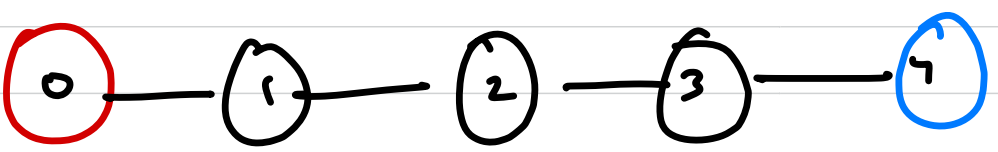
O(V!)

Q=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.

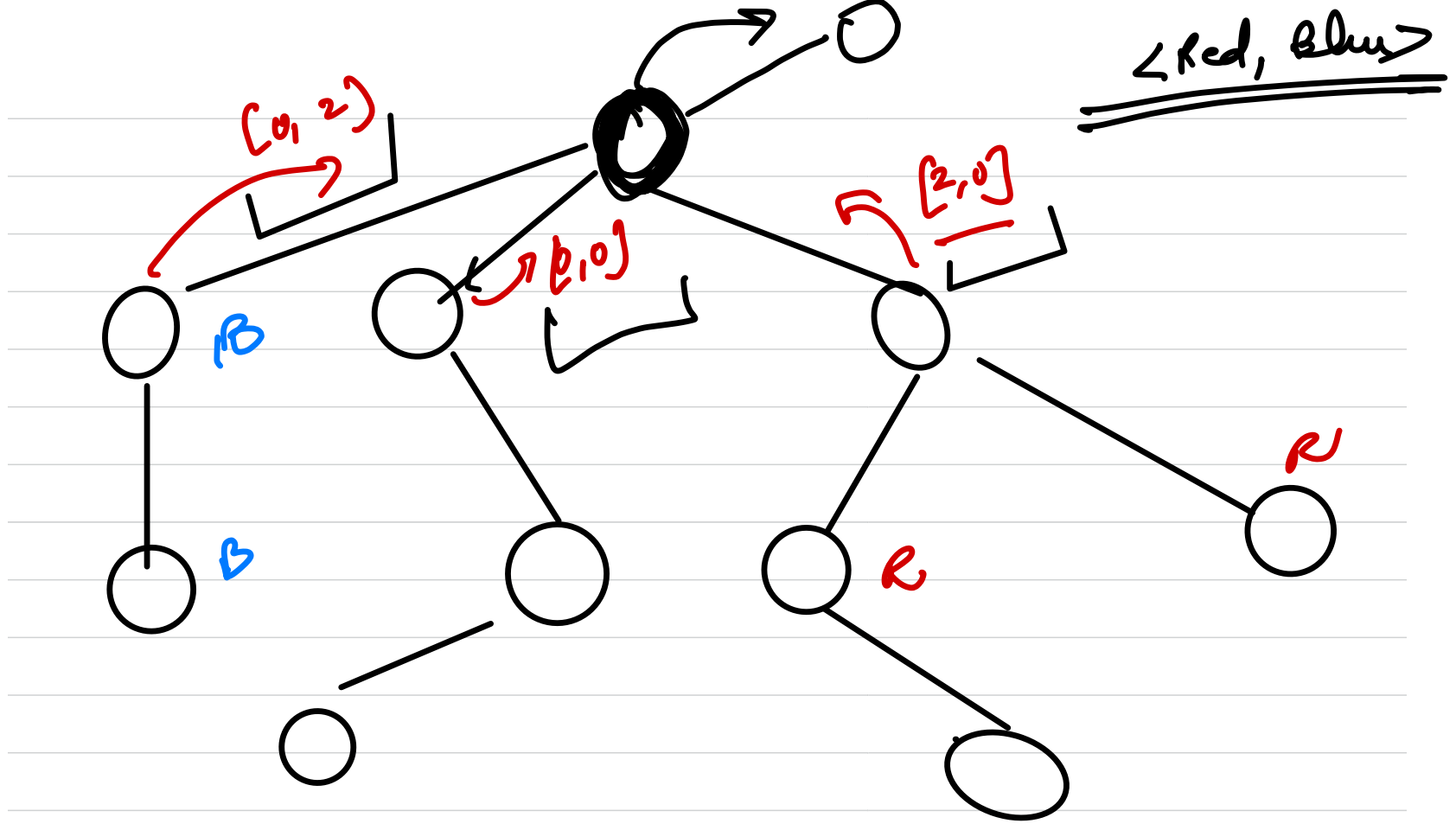
$v \leq 3 \times 10^5$



1 ano

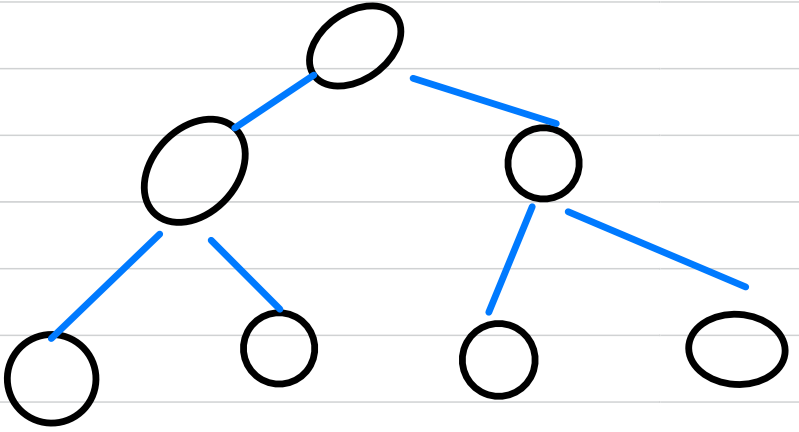


4


<Red, Blue>
[0, 2)
[2, 0]
[1, 0]
B
B
R
R

Tree → V nodes

edges ??



Prove that a tree with n nodes has n-1 edges.

PMI → assume $f(n) \rightarrow n-1$

$f(n)$ → no. of edges for n nodes.

$f(1) \rightarrow 0$

$f(2) \rightarrow 1$

Base Case

assume $f(n) \rightarrow$ true

To prove $f(n+1)$

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

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

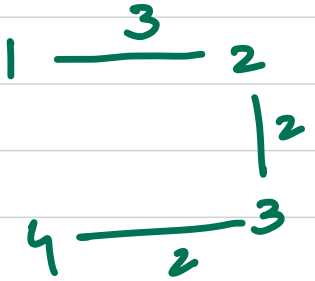
$$f(n+1) = f(n) + 1$$
$$= (n-1) + 1 \Rightarrow (n+1) - 1$$

$f(n) = n-1$ H.P.

→ Holi (Spoj)

1 → 3+2 → 5
3 → 2+3 → 5
4 → 2+2 → 4
2 → 2+2 → 4

18 ← belta



1 ⊂ 2
4 ⊂ 3

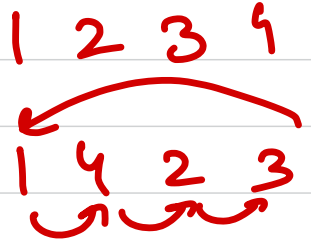
1 → 3
2 → 3
4 — 2
3 — 2

10

Brute force

1. 2. 3. ..... N → We can have N!

permutations



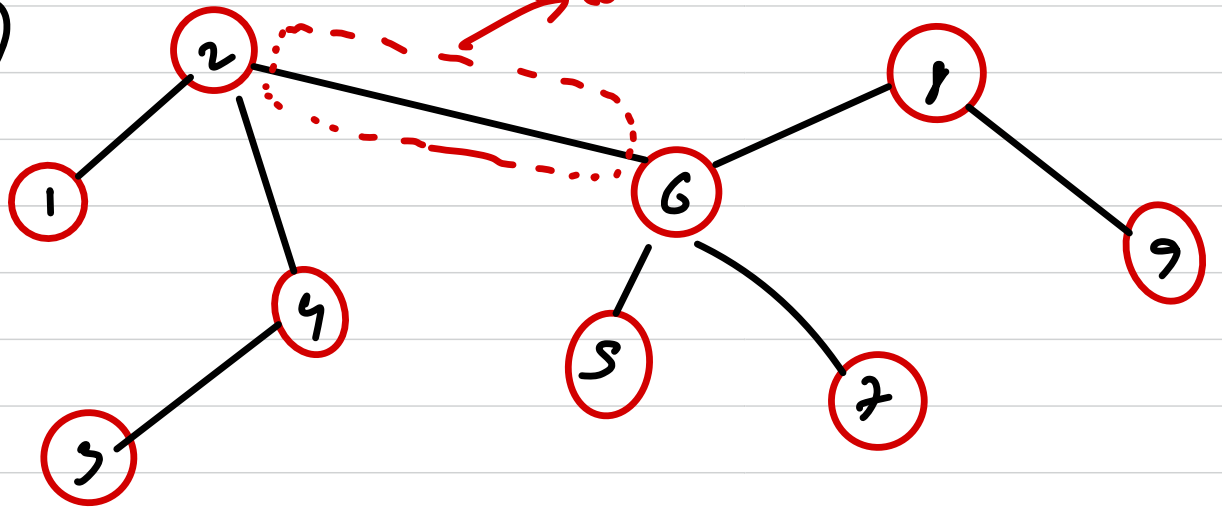
2-7

N nodes → (N-1) edges

1 — 4

Tree (weighted)

max combination of each edge



E × O(V+E)

V E + E²

O(V⁴)

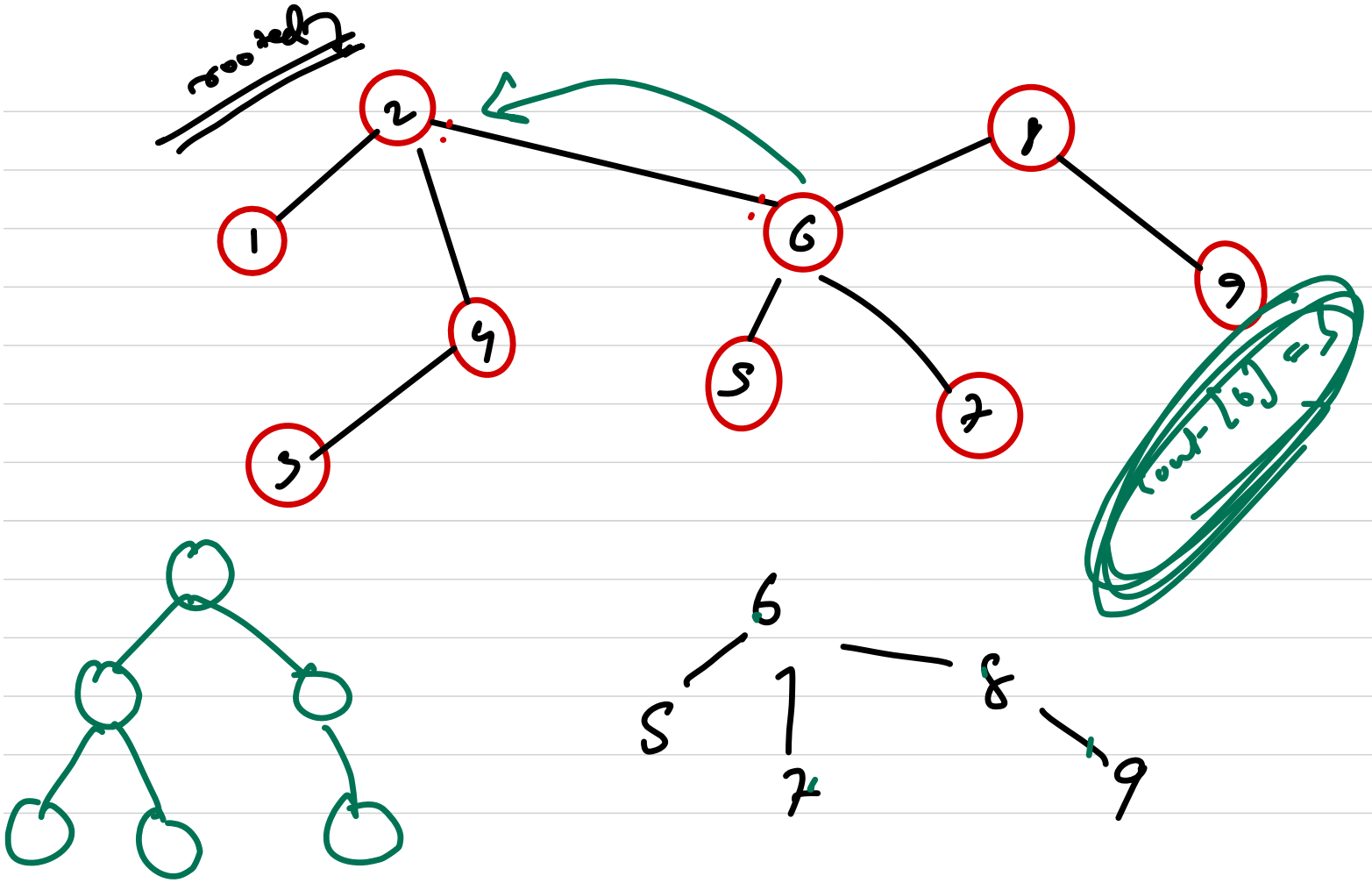
T(E)

1 — 5 5 — 2

2 — 6 6 — 3 . . . .

For any edge e,

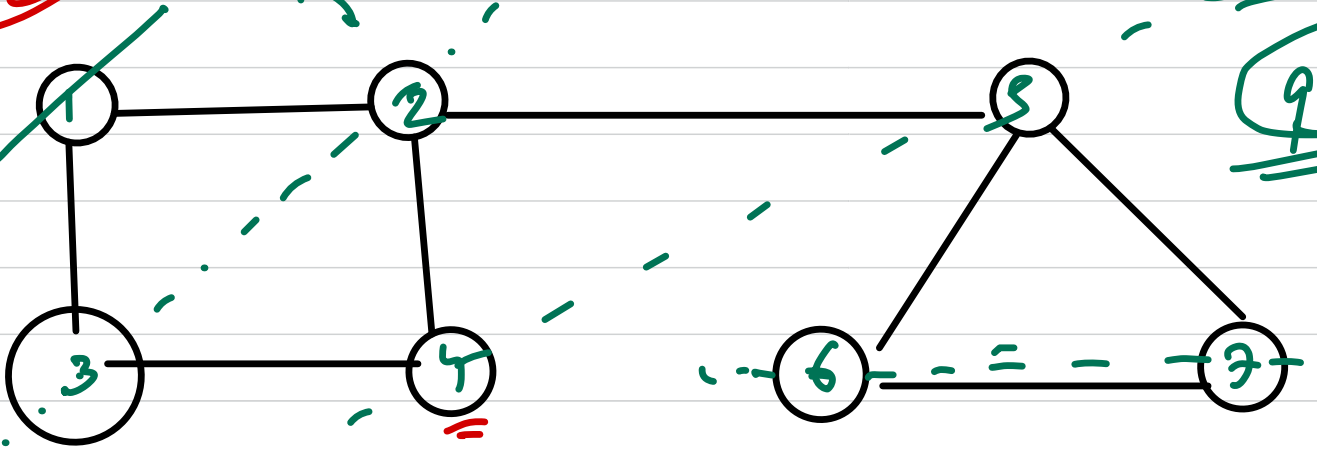
Contribution = $2 \times wt \times \min(s_2 - c_1, s_2 - c_2)$


root node
2
1
4
3
6
5
7
1
9
cout << [o] = 1;
6
5
1
7
8
9

vis

Breadth first Search

FCFS
FIFO
queue



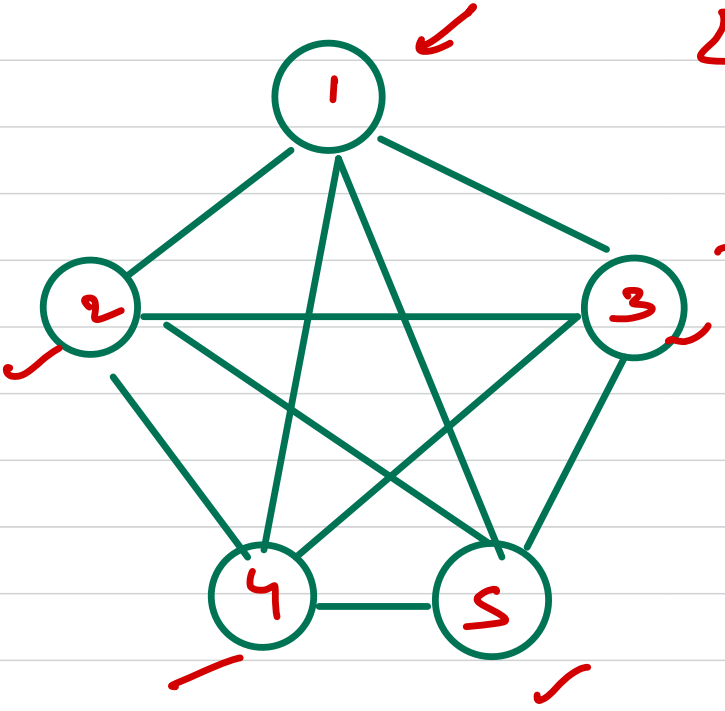
queue

1 2 3 4 5 6 7
x x x x x x x

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

level order traversal

TC → O(V+E)

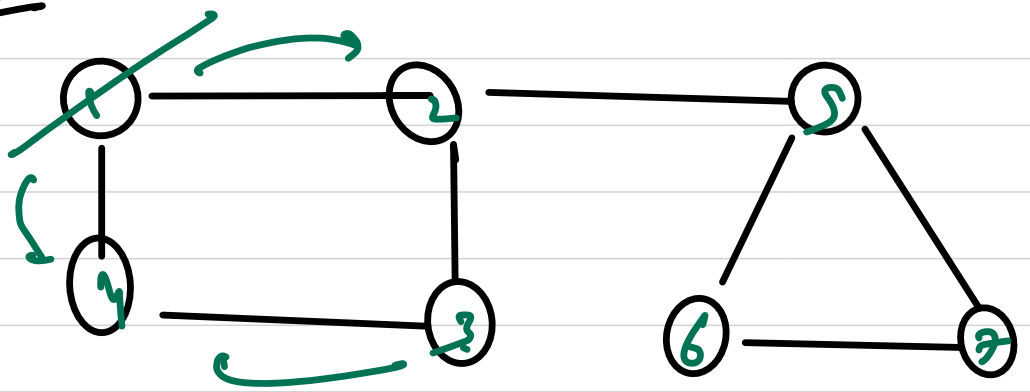


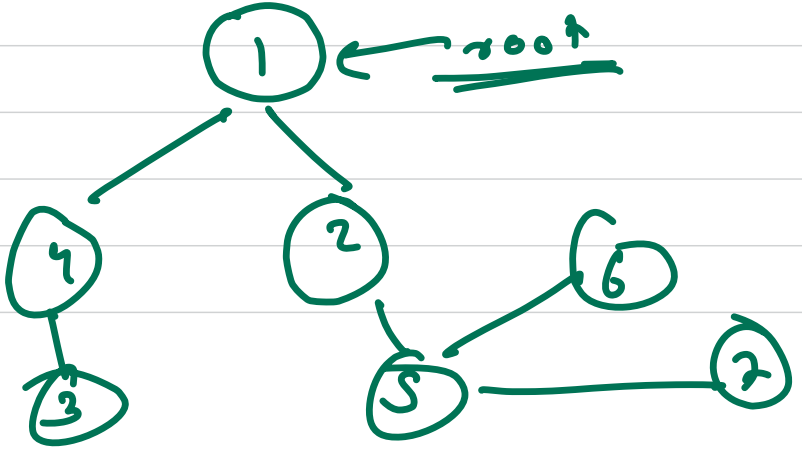
1 2 3 4 5

BFS Tree

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

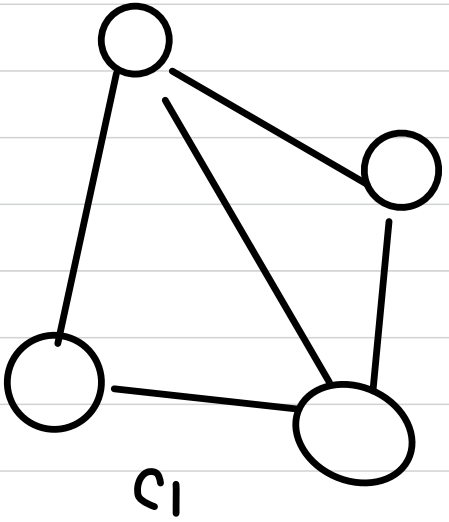
parent [, 1 , 1 , 2 , 2 , 5 , 5]



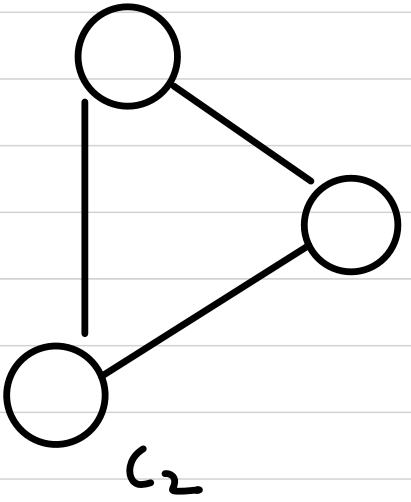


BFS Tree

★ Connected Components



C_1



C_2

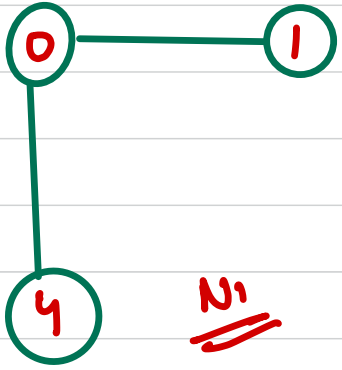
G

Journey to the moon

$^nC_2$ ① ② → Total

Total − Σ pairs of Some natural

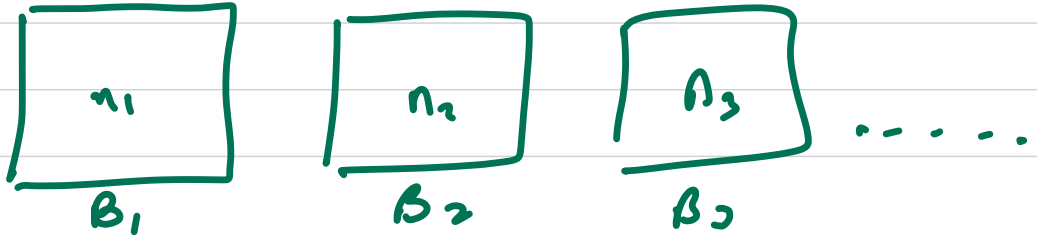
DFS



N_1

N_2

# of Sum of calls dfs = # of CC

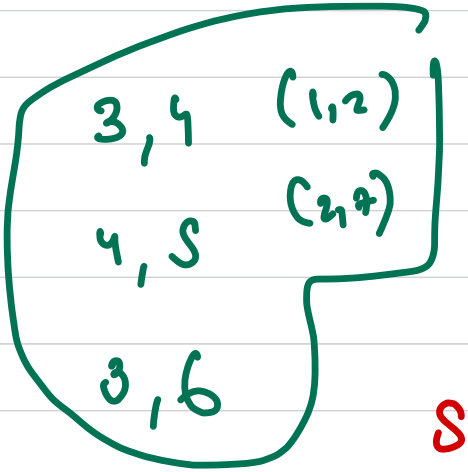


Q:- You have a permutation of N integers 1, 2, 3 .... N. You want to rearrange this permutation to some other permutation P. You can swap only M given pairs, whatever no. of times you want. Figure out if this is possible or not??

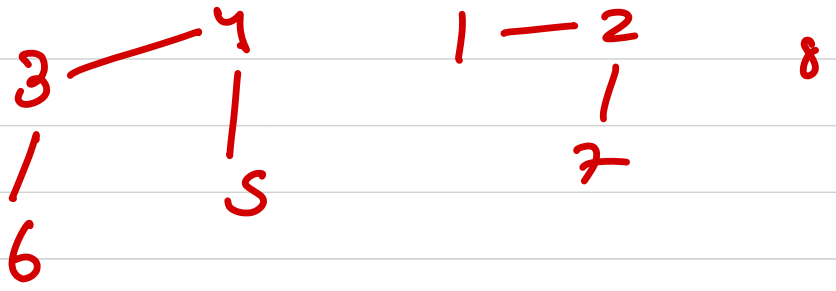
G → 1, 3, 2, 4 (3, 4)

P → 1, 4, 2, 3 → Yes

→ connected component ← direct application



indexes



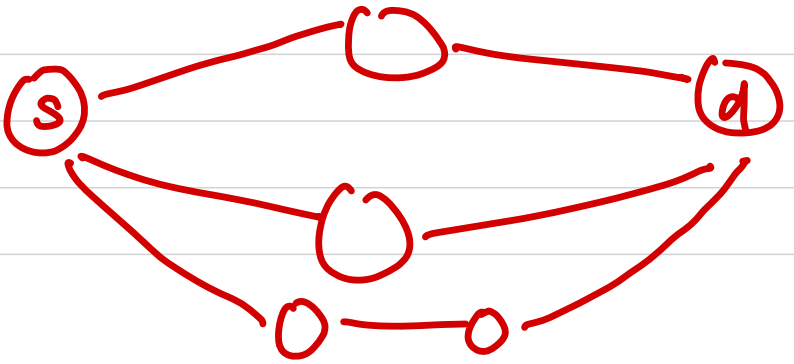
Set → G → [3, 4, 5, 6]

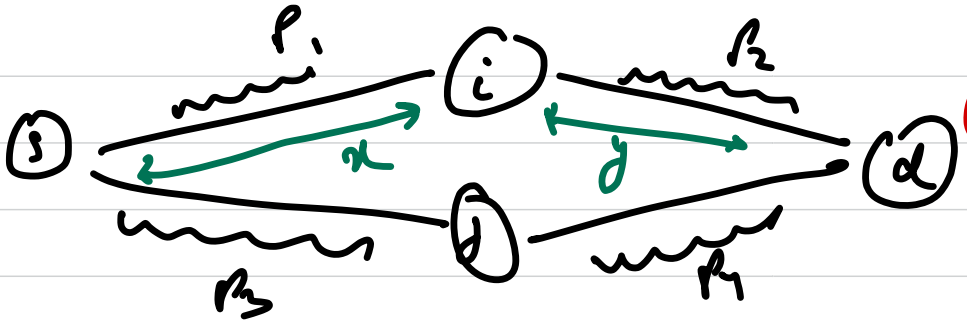
G →	1	2	3	4	5	6	7	8
P →	7	1	4	6	5	3	2	8

all the elements in one connected component can be arranged in any permutation under some swaps

→ BFS → Graph → Shortest path

Q) Given a graph, given src & dest nodes, find all the nodes which are part of atleast 1 shortest path.





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

$O(V+E)$

$y = d - x$

$x - y = d$

Disjoint Set Union

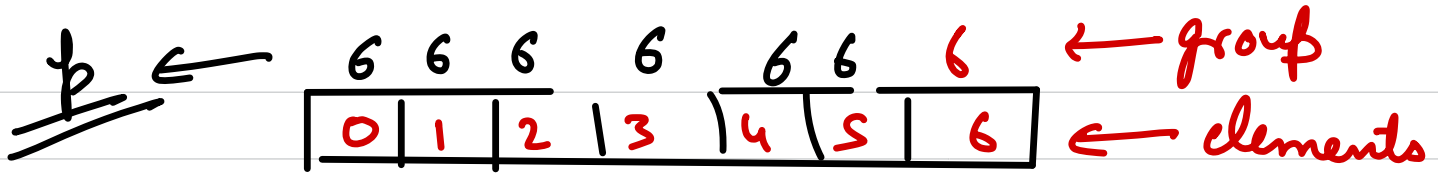
↳ We want to create clusters

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

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

1) union(a, b) → adds b to the group of a or vice-a-versa.

2) get(x) / find(x) → to what group / cluster x belongs to.



union(0,1)

union(2,3)

union(2,4)

union(2,5)

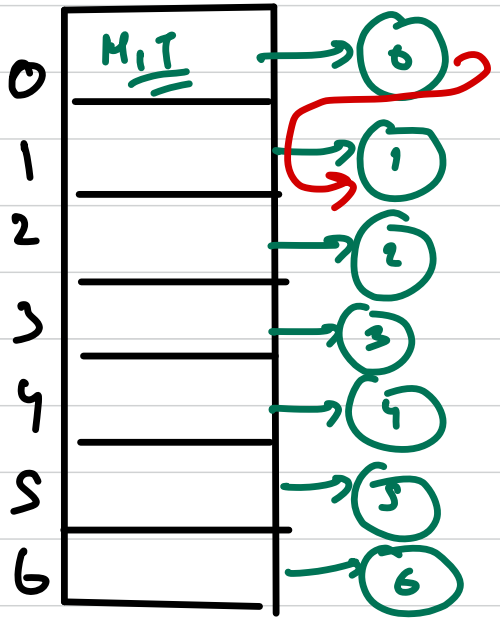
union(2,0)

union(6,2)

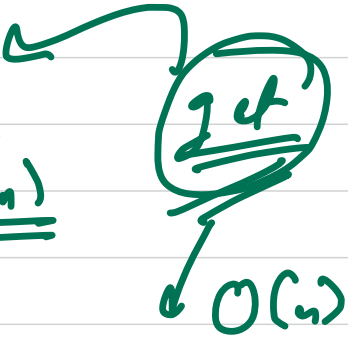
```
int get(int x) {          // O(1)
    return p[x];
}

void union(int a, int b) {   // O(n)
    a = get(a);
    b = get(b);
    for i in 1..n {          → linear search
        if p[i] == b
            p[i] = a;
    }
}
```

// How about storing groups as ll. / dll



union
O(n)

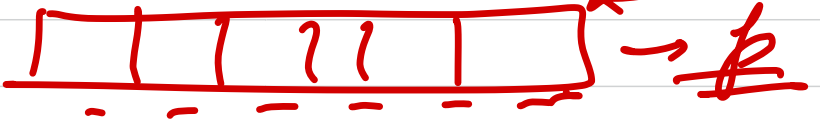


2 ⇄ 4 ⇄ 5 ⇄

O(n)

O(n)

O(1)

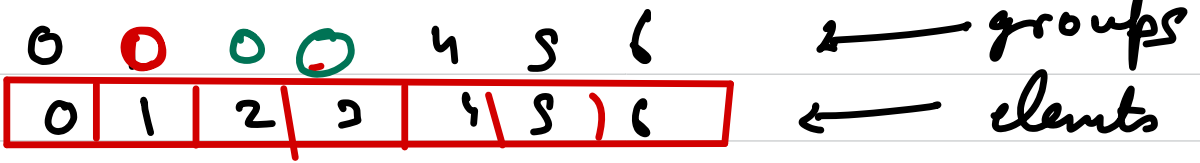


// So what we can see, if we add n elements then the operation of updating parent array for each element is $O(n)$

amortized $\quad \frac{n \times O(n)}{n} \rightarrow O(n)$

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

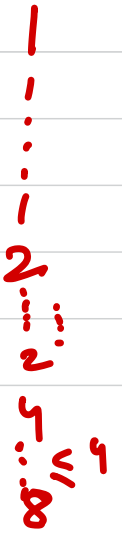
union by size

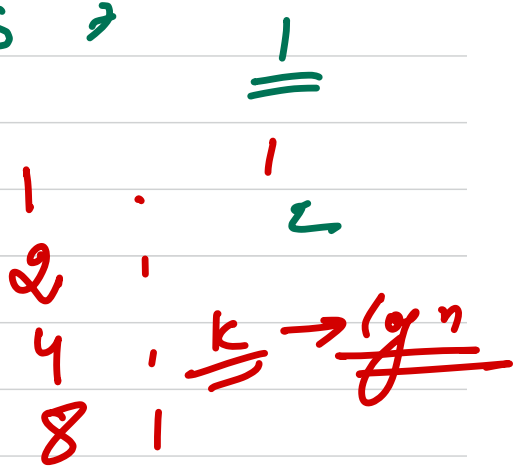


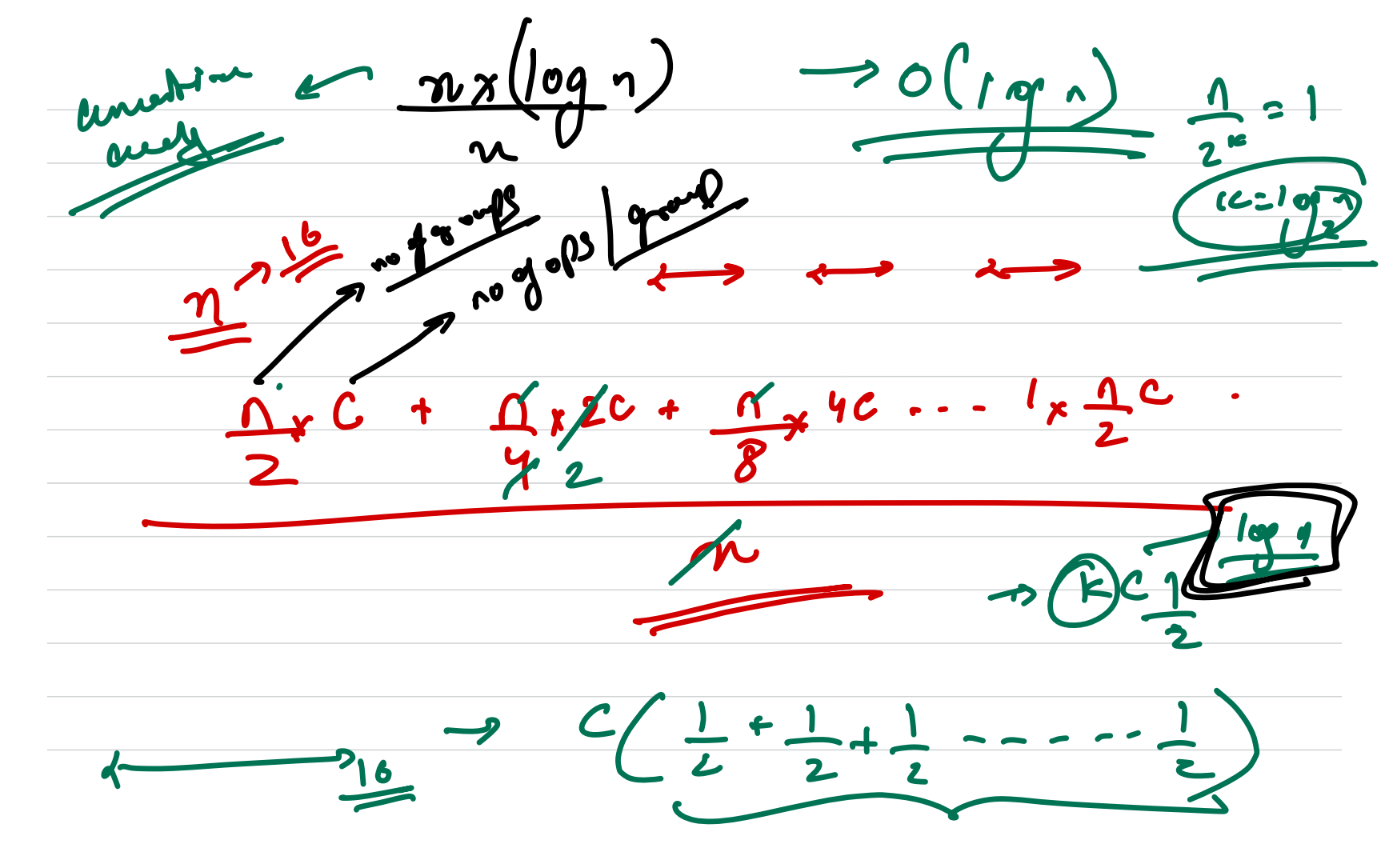
union (0,1)

union (2,3)

union (4,0)






$\frac{n \times (\log n)}{n}$
$O(\log n)$
$\frac{n}{2^k} = 1$
$k = \log n$
$n \to 16$
no of threads
no of ops / thread
$\frac{n}{2} \times C + \frac{n}{4} \times 2C + \frac{n}{8} \times 4C \cdots 1 \times \frac{n}{2} C$
n
$\log n$
$kC \frac{1}{2}$
16
$C\left(\frac{1}{2} + \frac{1}{2} + \frac{1}{2} \cdots \frac{1}{2}\right)$

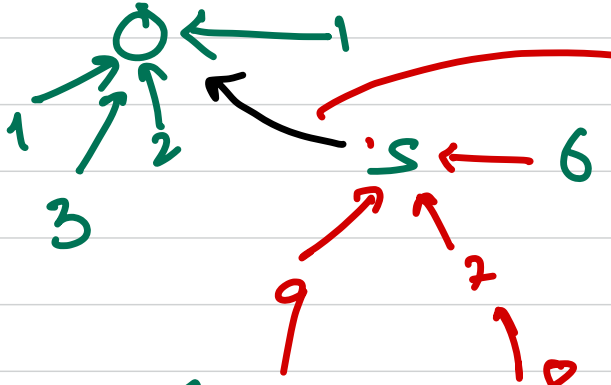
→ get → O(1)

→ min → O(log n)

Rank → level

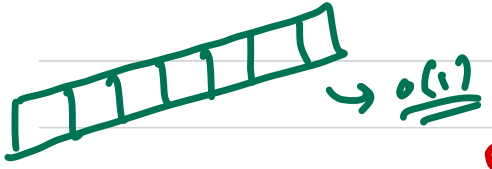
union by size

same size

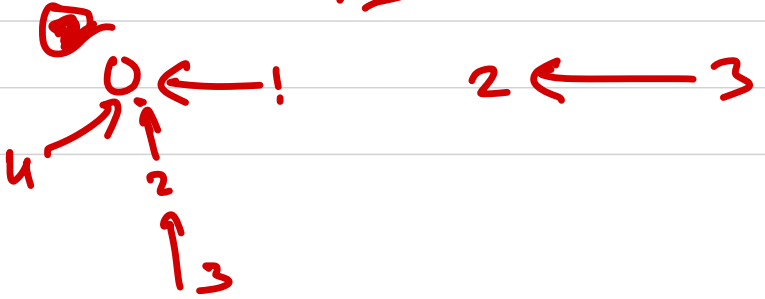


when is the new link added ?!

on size of about same size



8



log 1

path compression → inverse Ackermann → complexity

$O(\log^* n)$ → extremely slow growing func^n

$\log^* n$ → no. of steps we need to take $\log_2 n$ as the value n to make it smaller than one.

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

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

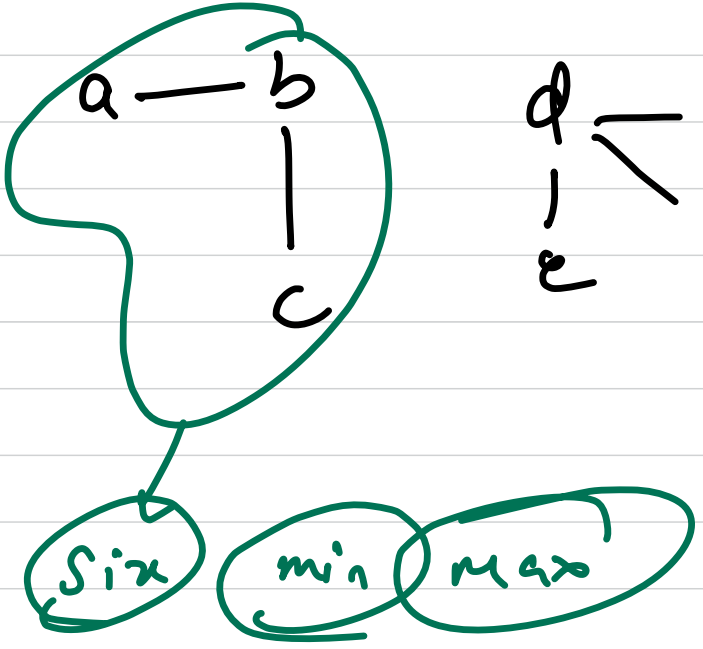
$\log 2^{16} \rightarrow \log 2^{4} \rightarrow \log 2^{2} \rightarrow \log 2^{1} \rightarrow 0$

1

connected component

$\longrightarrow O(V+E)$

$\hookrightarrow O(\log^* V)$



diamond
disjoint

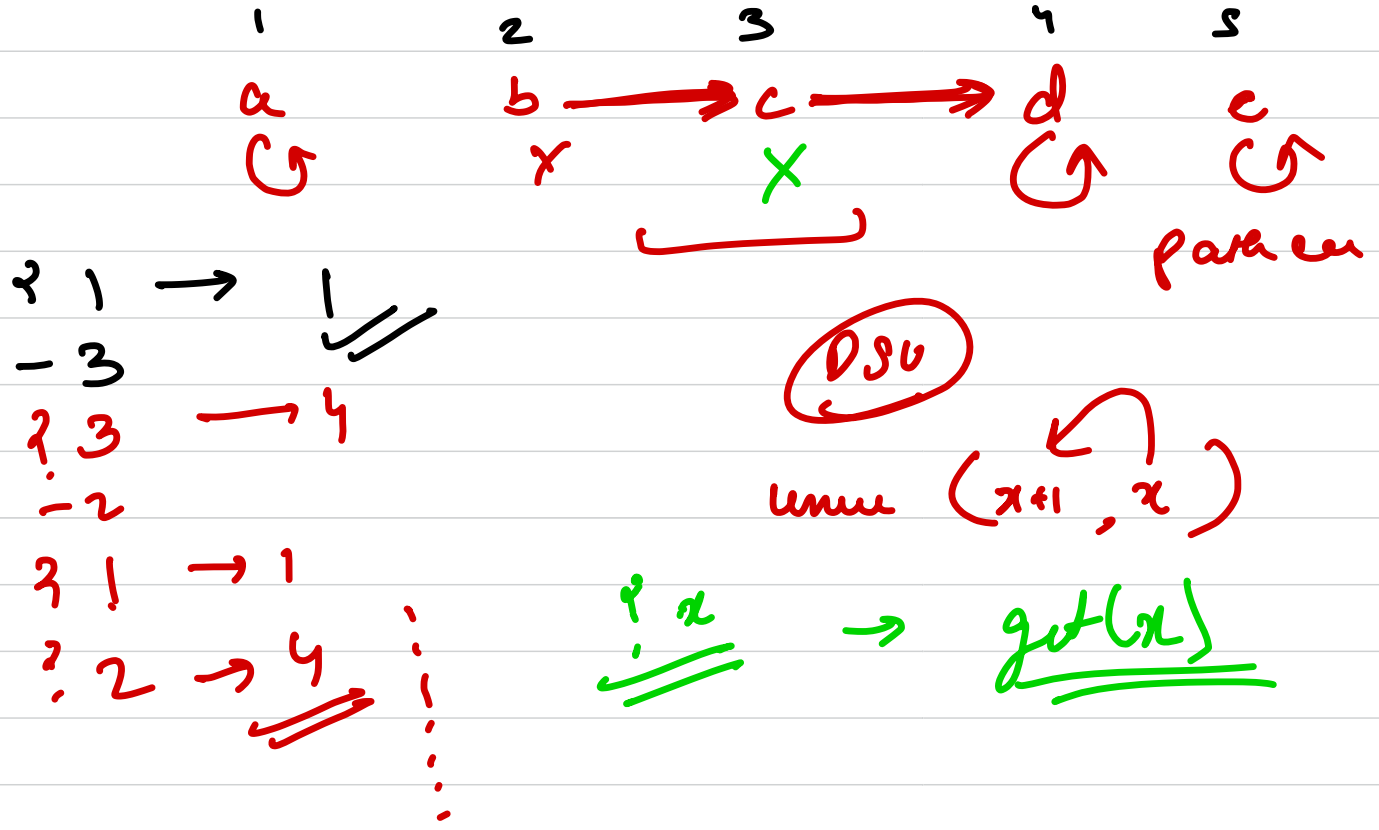
Q:) n persons standing in a row at positions [1-n]. We can do ops,

(1) - x → remove the person at pos x.

(2) ? x → find the nearest person to the right of x that is still present (not removed).

$n \leq 10^6$

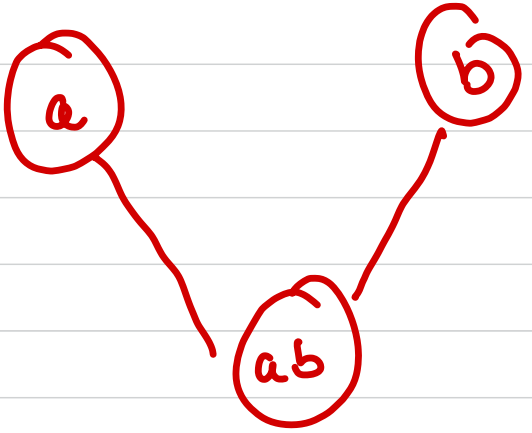
$q \leq 10^6$


1
2
3
4
5
a
b
c
d
e
x
X
parent
DSU
union (x+1, x)
? 1 → 1
- 3
? 3 → 4
- 2
? 1 → 1
? 2 → 4
? x
get(x)

Q21 Secret Password - Codeforces

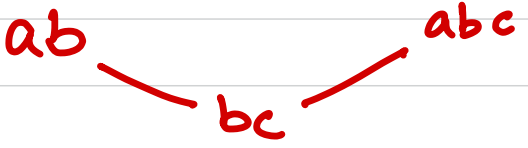
↳ DSU ?

DFS



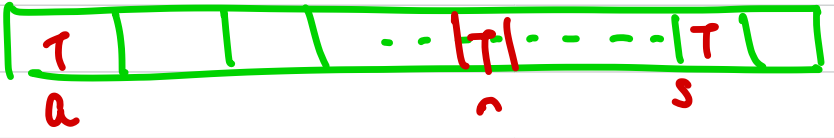
d

Graph Components

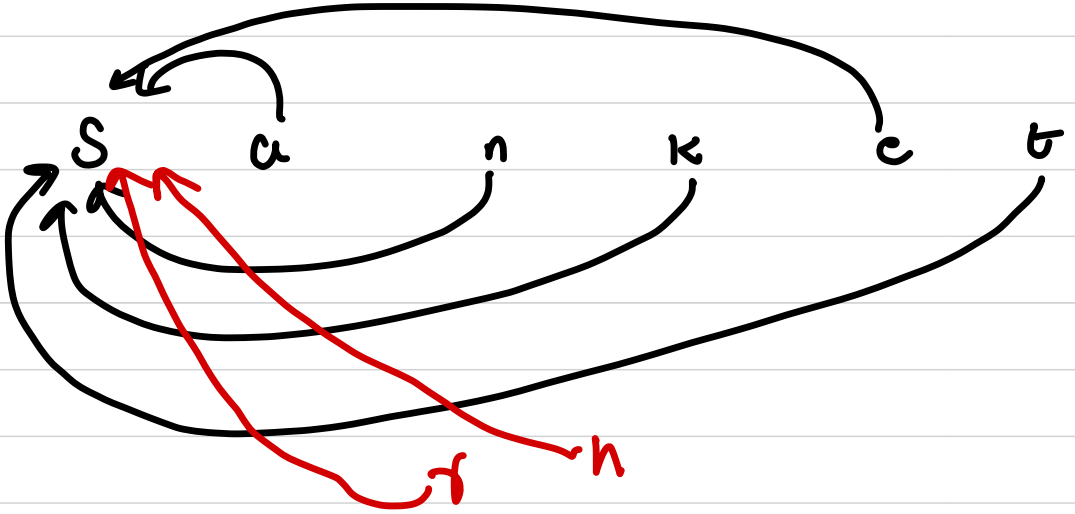


(1)

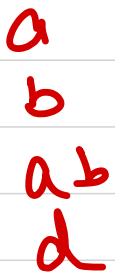
26

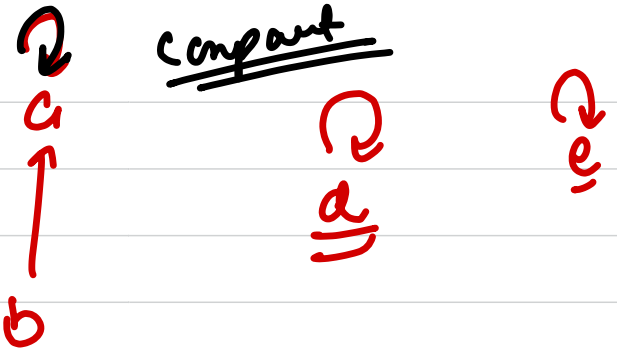


Sanket



as that





Total nodes → 26

1 node → 1 char

# of nodes parent[i] == i → # of compact.

Q: Given an undirected graph. Some ops are given,

remove a, b → remove edge from a, b

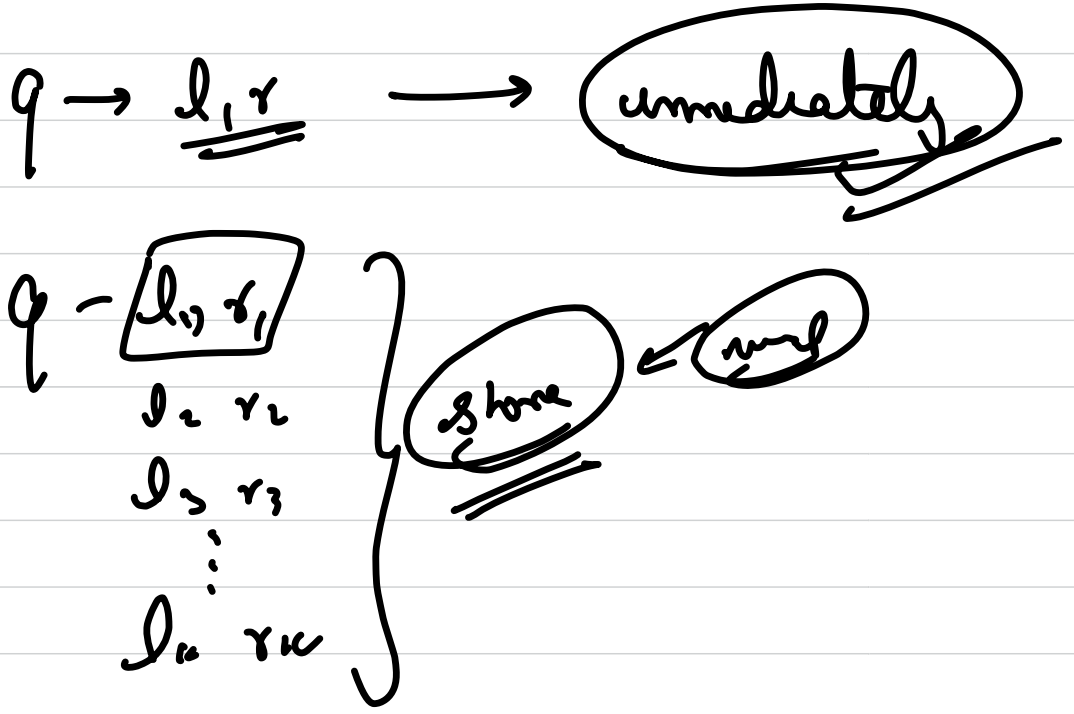
get a, b → tell if a & b are in same component

$V \leq 5 \times 10^4$
$e \leq 10^3$
$q \leq 15 \times 10^4$

2

after all removal there is no edge left

Online query vs Offline query



⇒ DSU → union - find

offlinequery

2

c, c → T

a b → T

- a b

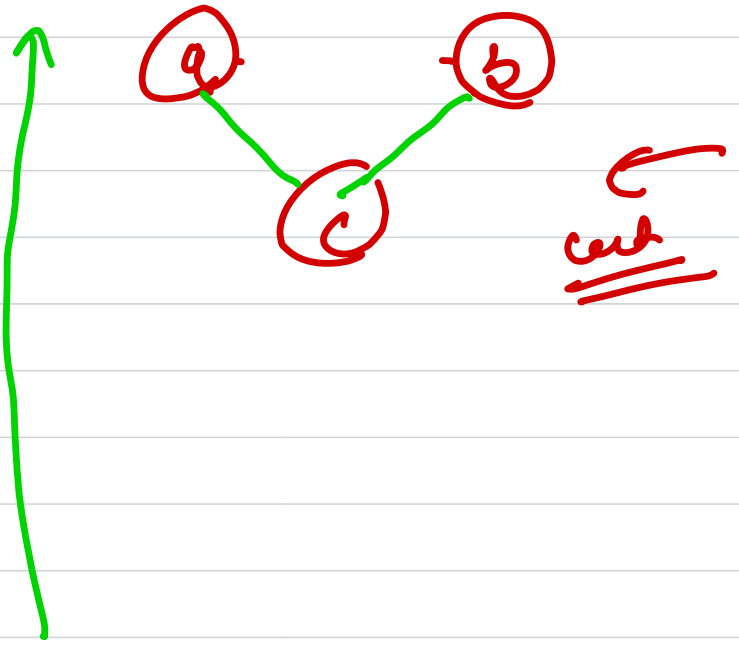
a, b → T

- a, c

b, a → f

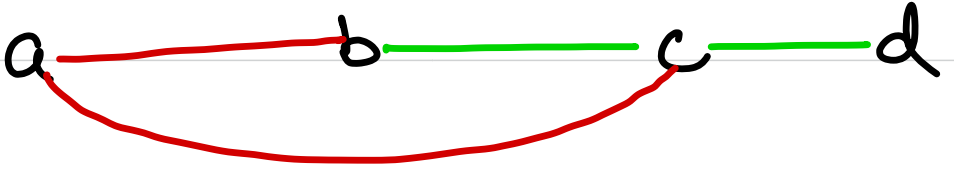
- b, c

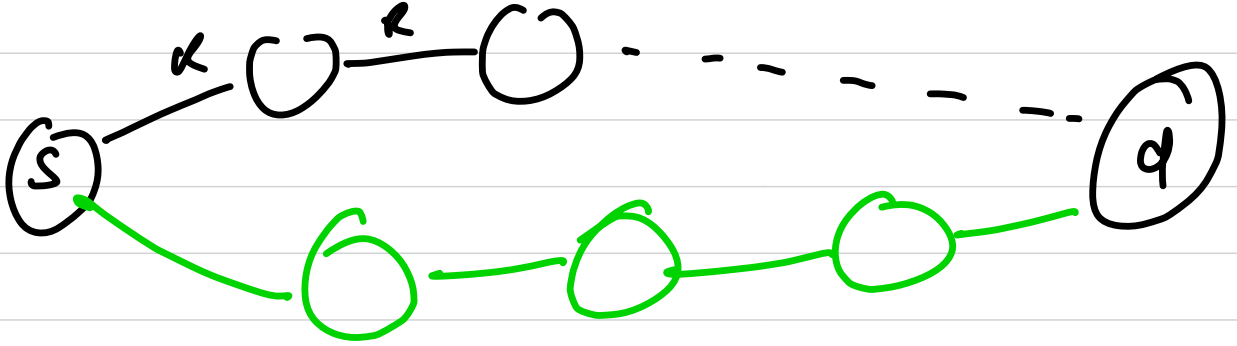
a b → f



end

Q: Given a graph with V nodes & E edges, every edge has an associated color to it with 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 once.




x
x
s
q

Q:- 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.

["wrt", "wrf"]

↳ wrtf

a b c

a < b < c

a < d < e

b < d.

b < c

d < e

b < e

a d e

inequalities

Dependencies

Q ⇒ Alien-Dictionary → Leetcode

english → 'a' ← starts with

[wrt , wrf , er , ett , rftt] → f, t

w, w, e, e, r

w, e, r ⇒ w < e < r

wrt wrf ⇒ t < f

r < t

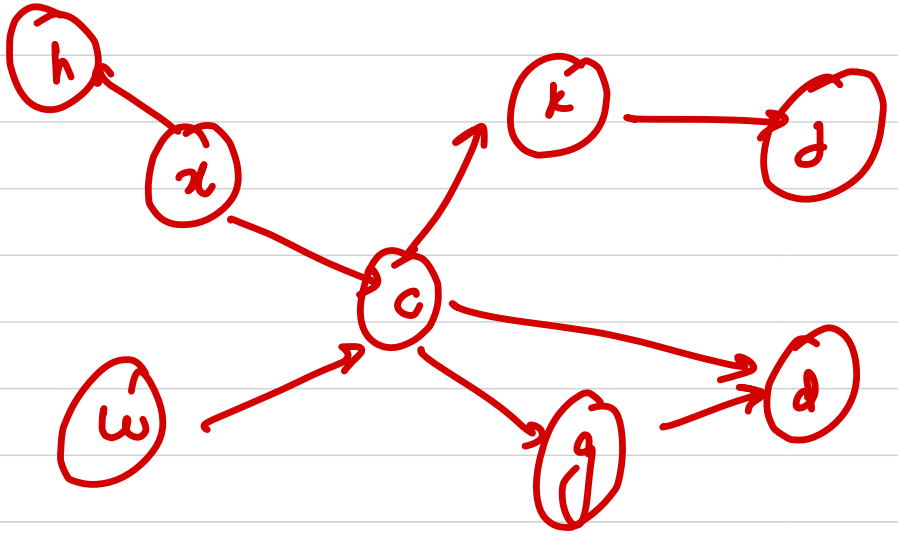
→ dependency graph

$w < e < r$

character

$w \longrightarrow e \longrightarrow r \longrightarrow t \longrightarrow f$

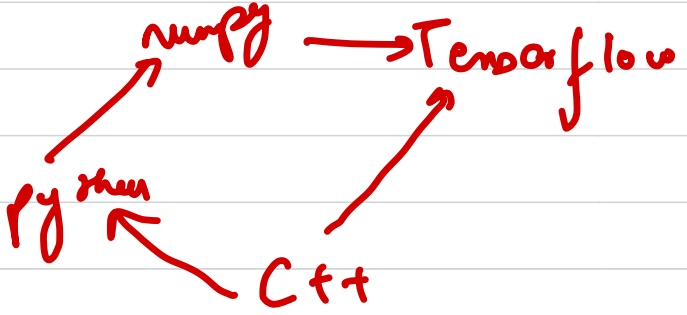
$w \; e \; r \; t \; f$

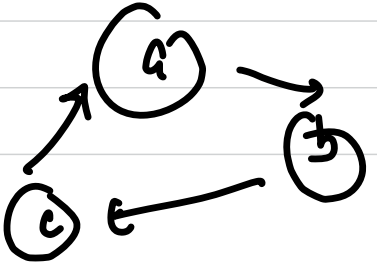


Complex dependency graph → a dep of cluts

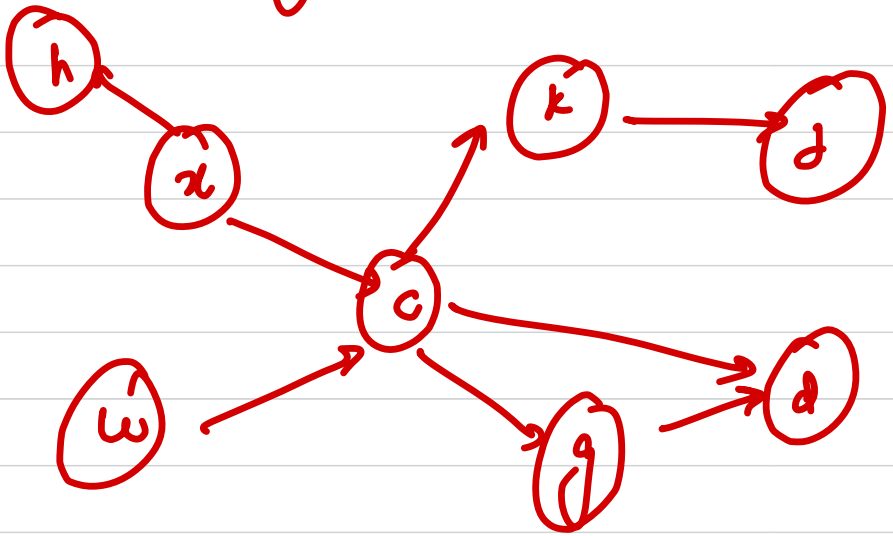
Topological Sorting

C++, Python, numpy, Tensorflow





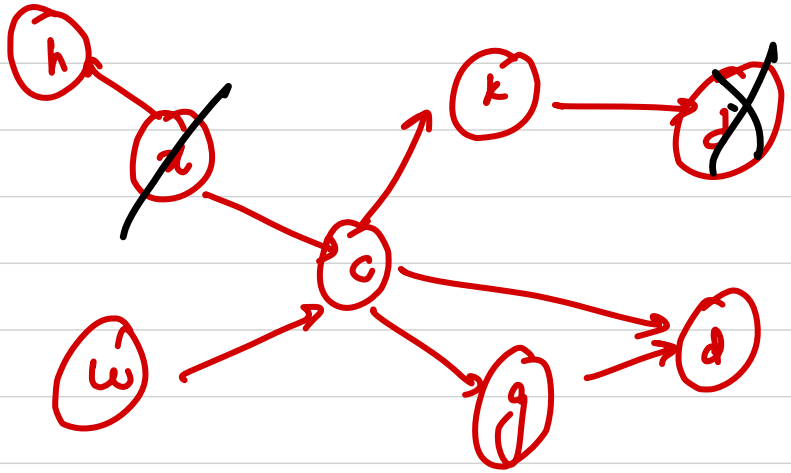
Kahn's Algorithm (modified bfs)



↙

[Directed Acyclic Graph]

[Resolve those first who have no dependency] ⟲ repeat



BF

$g[v].pb(v)$

$v \rightarrow v$

V<T

e

x, w, t, q, h, c, m, k, g, j

d

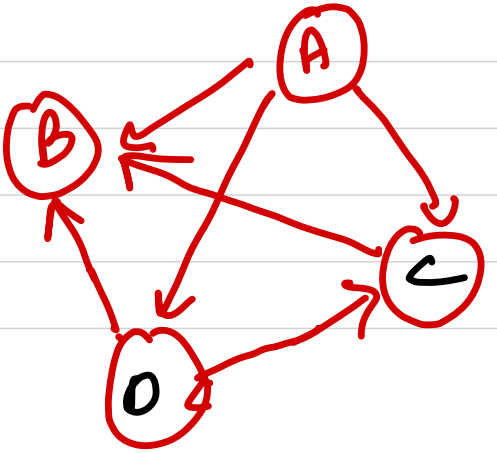
q → m

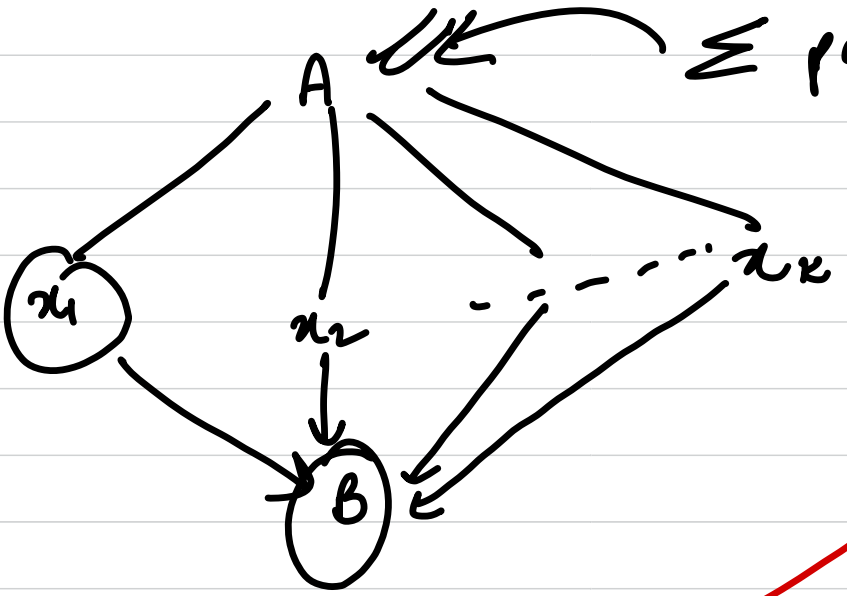
undirected → x, h, t, c, w, k, j, d, g, m, q

0, 1, 0, 2, 0, 1, 1, 2, 1, 1, 0

[ ]

Q» Given a directed acyclic graph with V vertices & 'e' edges. find the no. of paths that exists between any given node A to another node B.





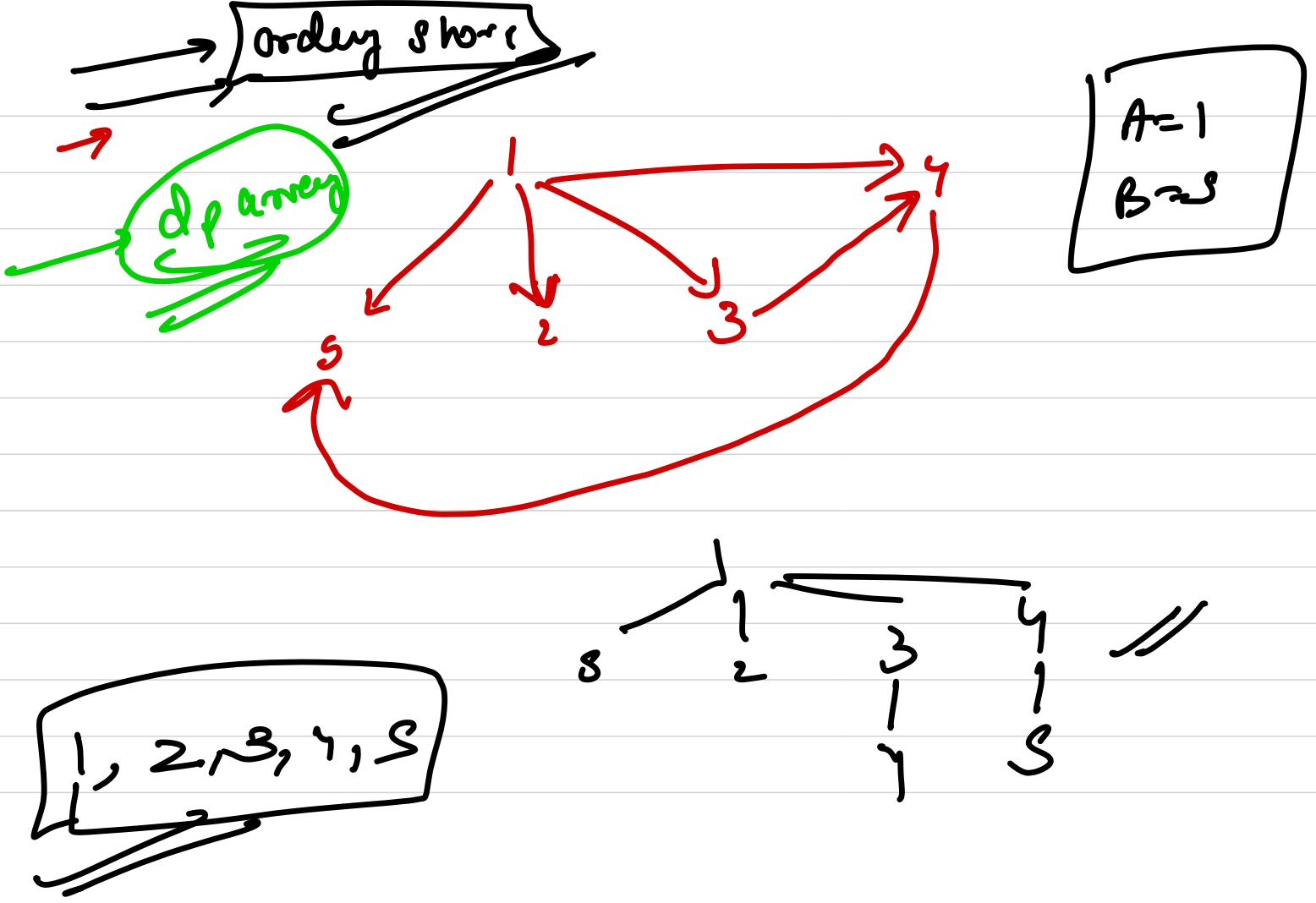
Σ paths(x_i, B)

$f(B,B) \to 1$

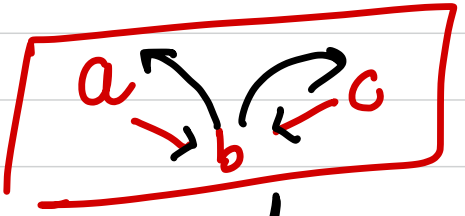
$f(v,v) = \Sigma f(x_i, v)$ $\quad x_i \in$ neighbours of v

$f(v,v)$: # of paths from v to v

$f(a,b) \to$ ans


ordering short
dp array
1
2
3
4
5
A=1
B=3
1, 2, 3, 4, 5

Strongly Connected Component.

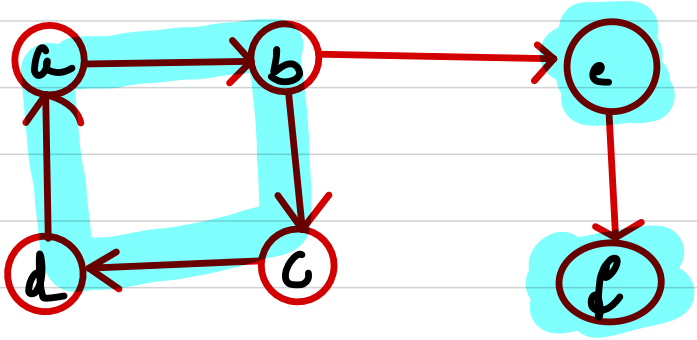


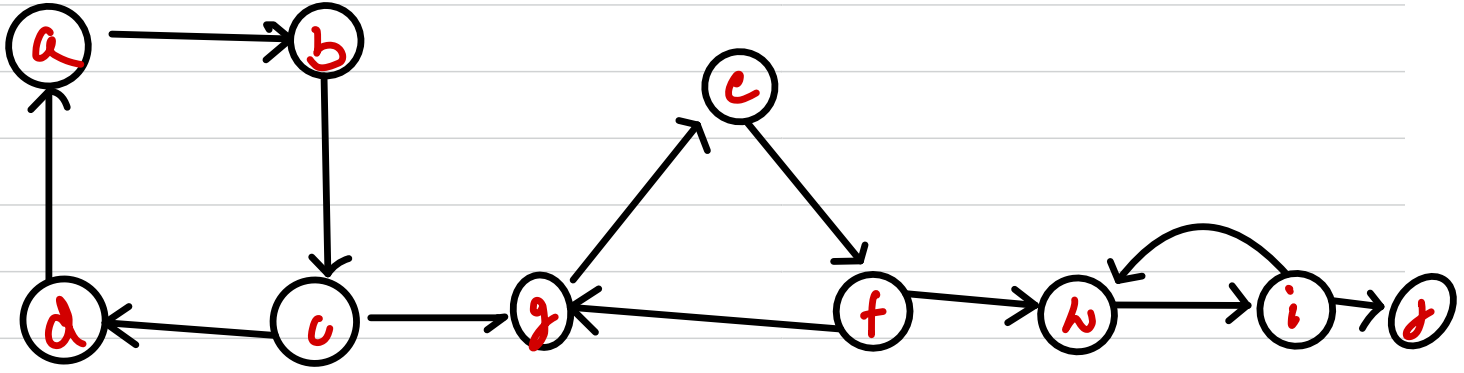
= cc

↓

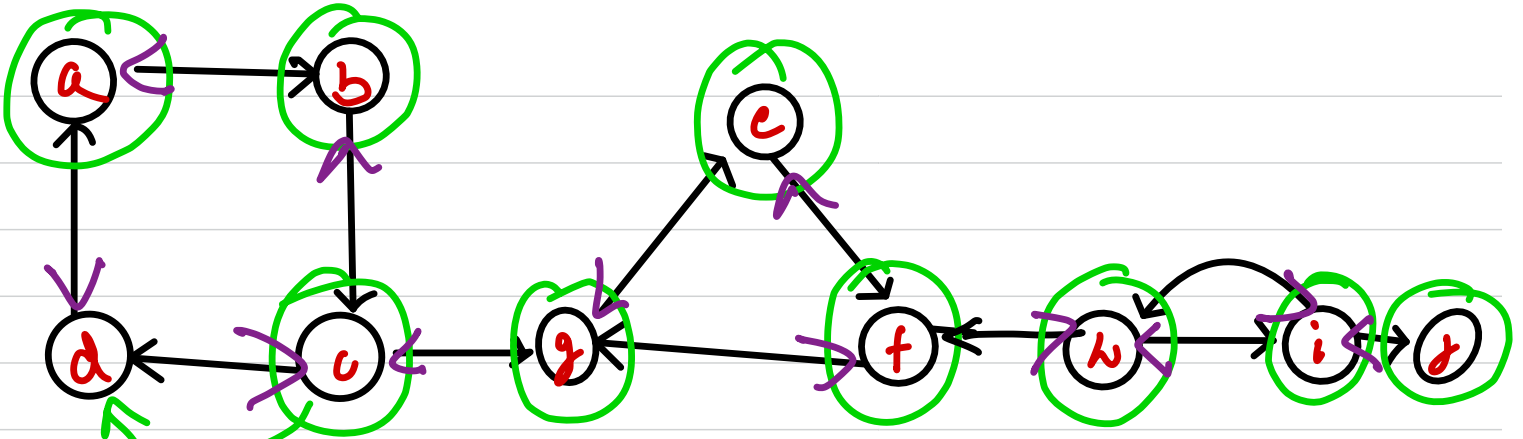
scc

How many strongly connected components are...





Kosaraju Algorithm → Used to find SCC



2

d | f | e | g | c | b | a | d | i | h

stack

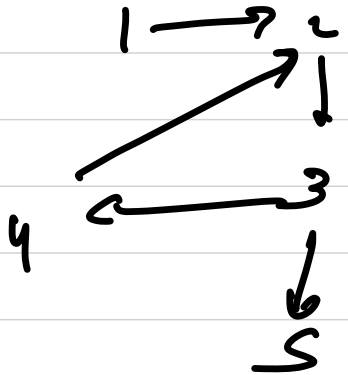
structure

perform DFS, & store nodes when they don't have unvisited neighbours

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

vis

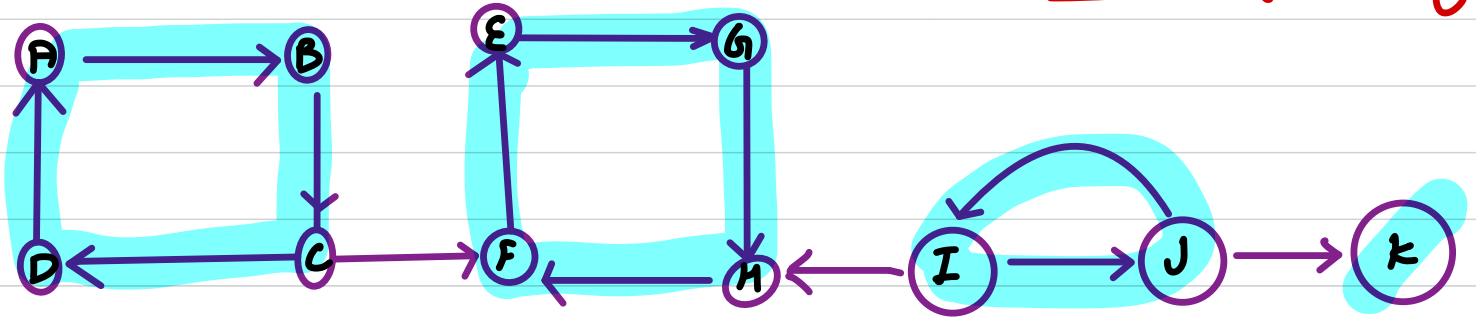
Transpose the graph



1, 2, 3, 5, 4

Strongly Connected Component

Kosaraju's Algo



$G \rightarrow G'$

Transpose

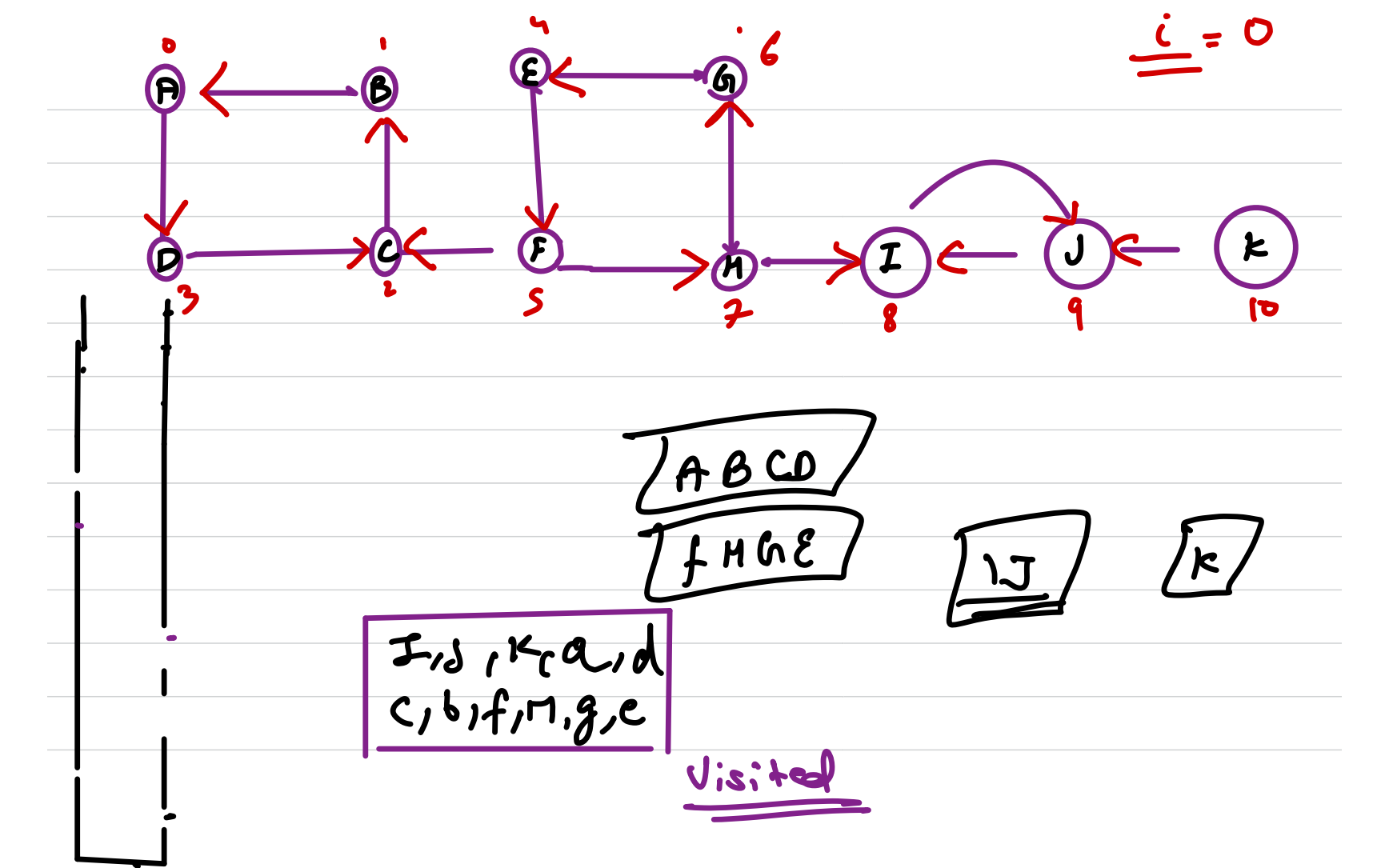
$G \rightarrow a \rightarrow b$

$G' \rightarrow a \leftarrow b$

Two phases

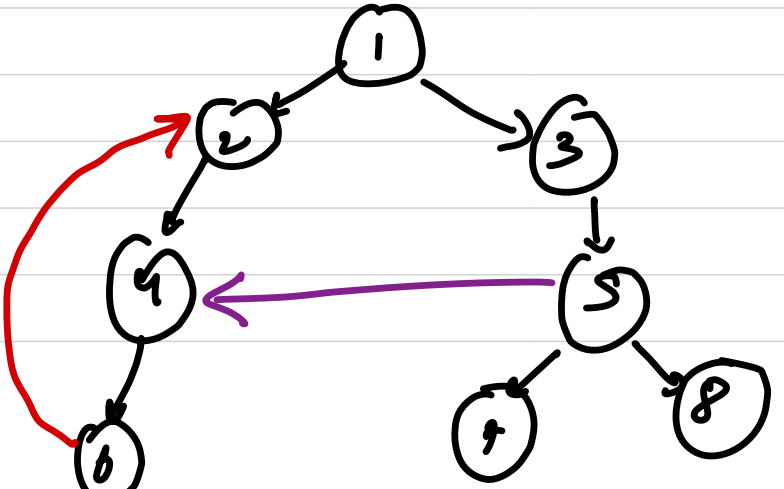
1) Before Transpose → we start a dfs, to find all vertices reachable from a node (v)

2) After transpose → we check the reverse i.e find the subset of nodes that can node (v)


i = 0
A
B
C
D
E
F
G
H
I
J
K
0
1
2
3
4
5
6
7
8
9
10
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. If we do dfs on G, we will get forest of trees

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

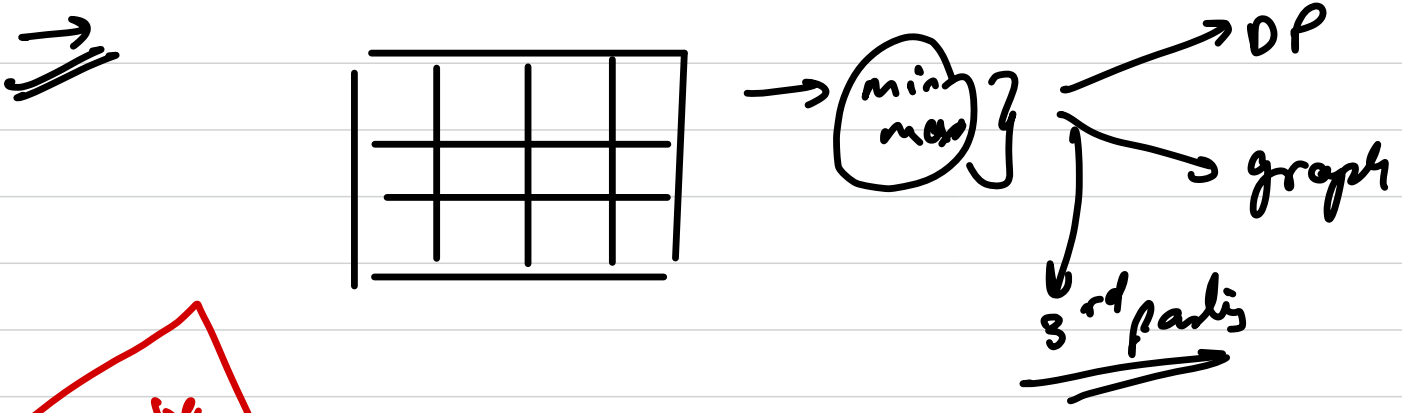


Back edge → It's an edge from (u, v) such that v is ancestor of node u but not part of dfs tree.

Cross Edge → It connects two node such that they do not have any ancestor & a descendant relation betwn them

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

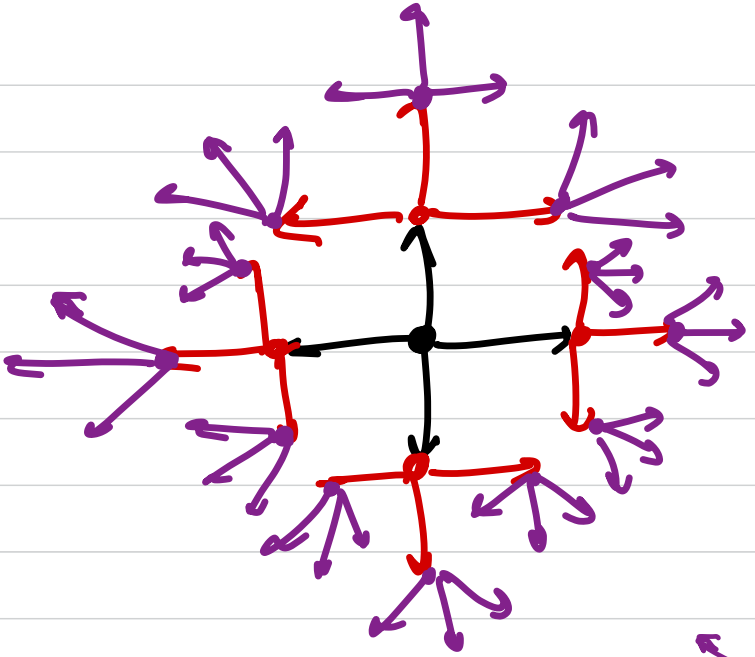
Suppose there is a cross edge from lower ordered nodes v_1 to a higher ordered node v_2. Then during the dfs on v_1, v_2 must have been visited. & we would be called dfs for v_2 in the ordering. Then in that case v_2's order is less.



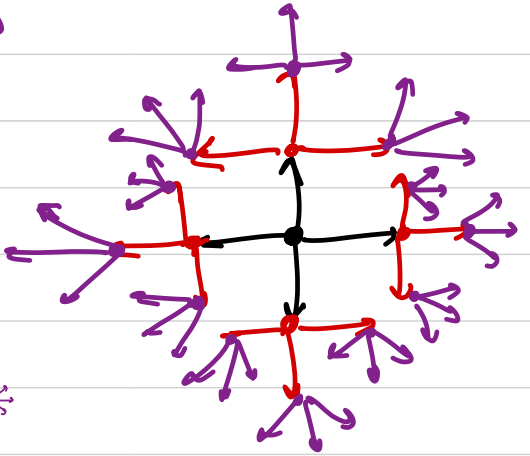
Can this be solved by Graph

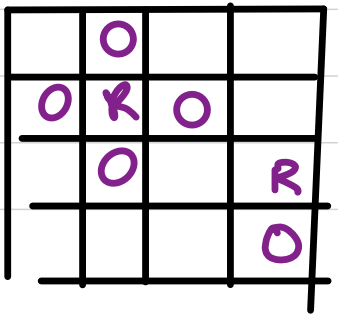
Multi Source BFS

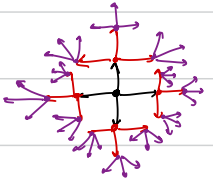
level 1 BFS

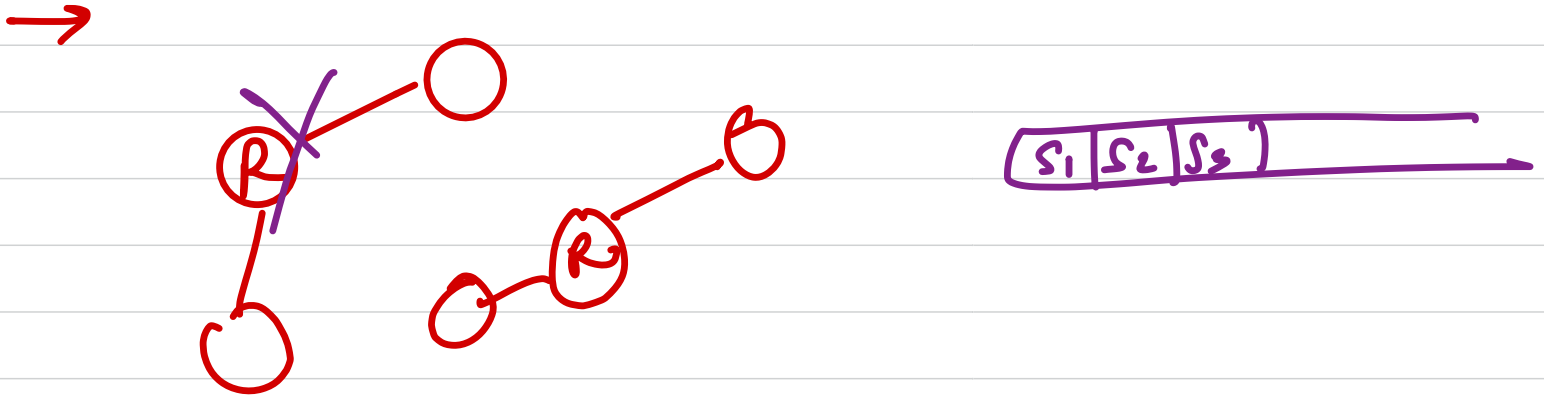


minimum time instance when last orange is get rotten

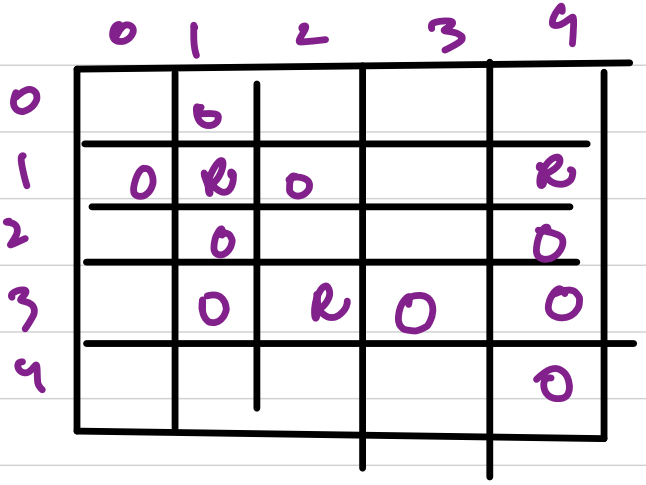








Push all the source in queue



Level order traversal

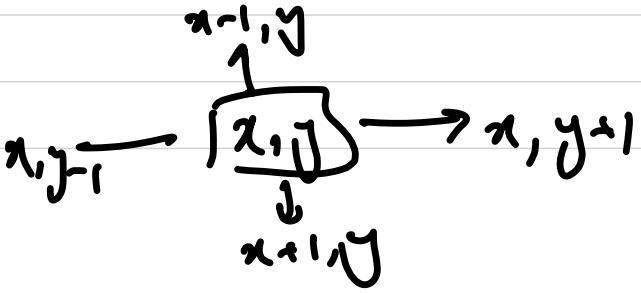
node → <x, y>

set< >

<x, y, d>

BFS

(1,1) (3,2) (1,4) (1,2) (0,1) (1,0) (2,1)



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 ?.

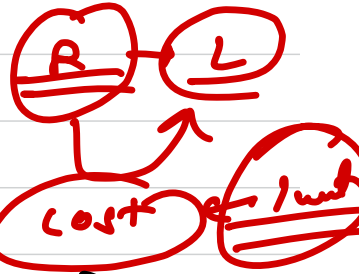
R	R	D
U	L	D
R	R	R

→ yes

swap

min cost

(0,0) → (n-1, m-1)



→ Bottom up code from (DP)

Worst Case → TC → O(nm)
SC → O(nm)

→ recursion

Top-down approach

R	R	D
D	L	L
R	R	U

Space - O(1)

worst case →
depth of path → nm

2 2

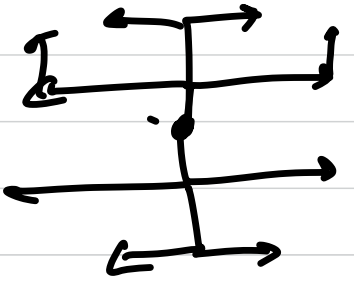
step counter

← BOS → LL → cycle detection

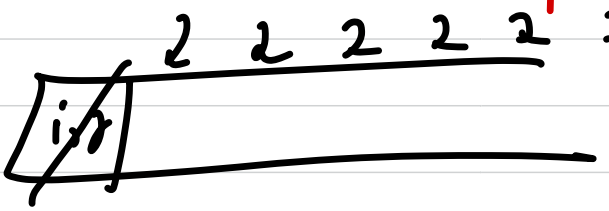
Knight on a chess board

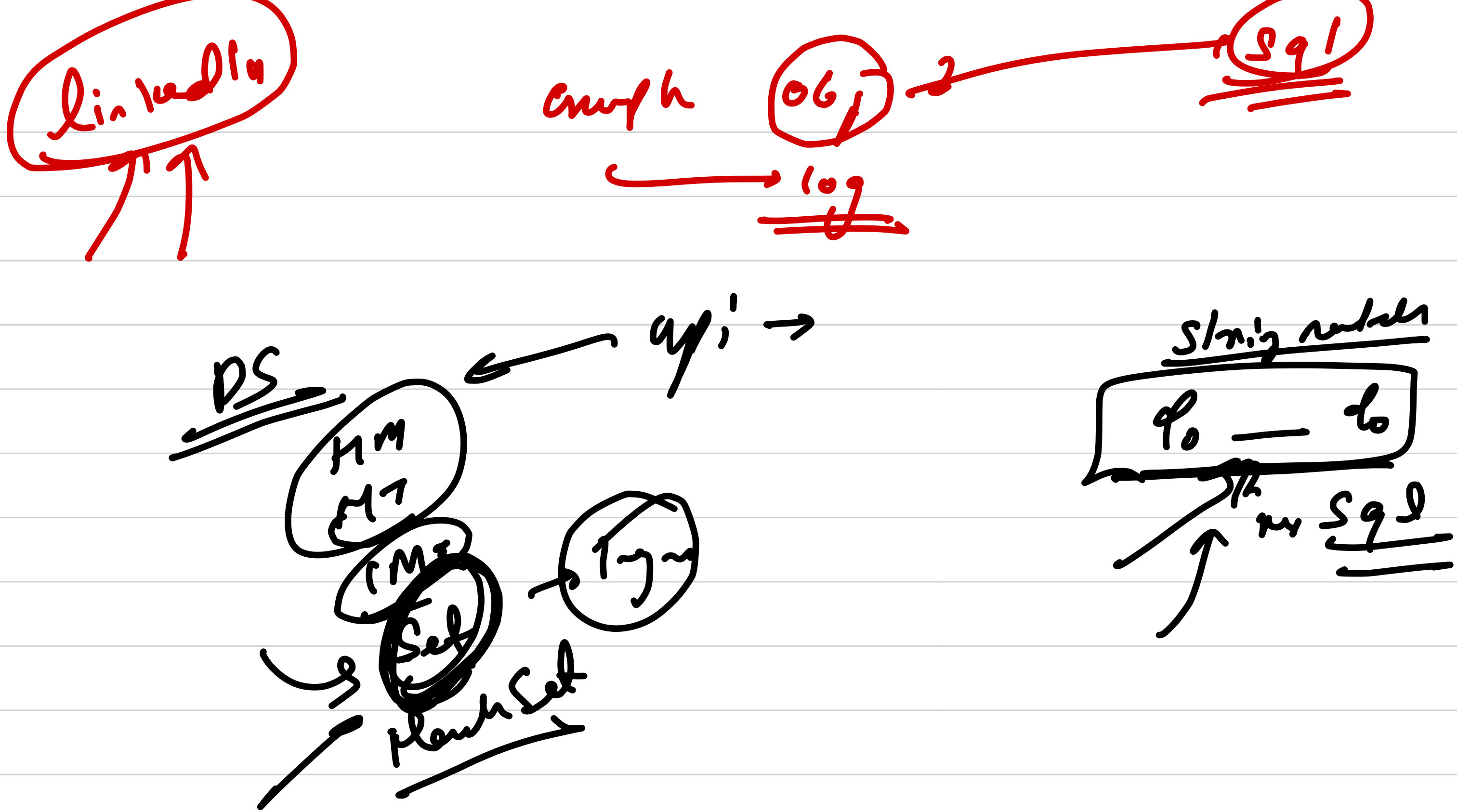
code 2.1

BFS



(i, j)





linkedIn
log
sql
api
DS
Set
HashSet
sql

Dun 20

Shortest path

	offset=0	offset=1	offset=2
offset=0	0 R	1 D	2 R
index=0	3 U	4 L (none)	5 D
2f5=0	6 R	R	D

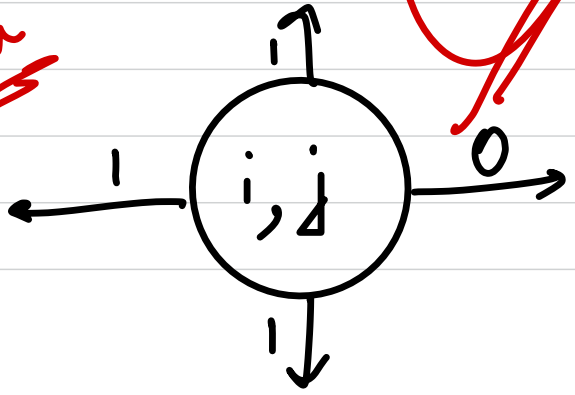
Hard

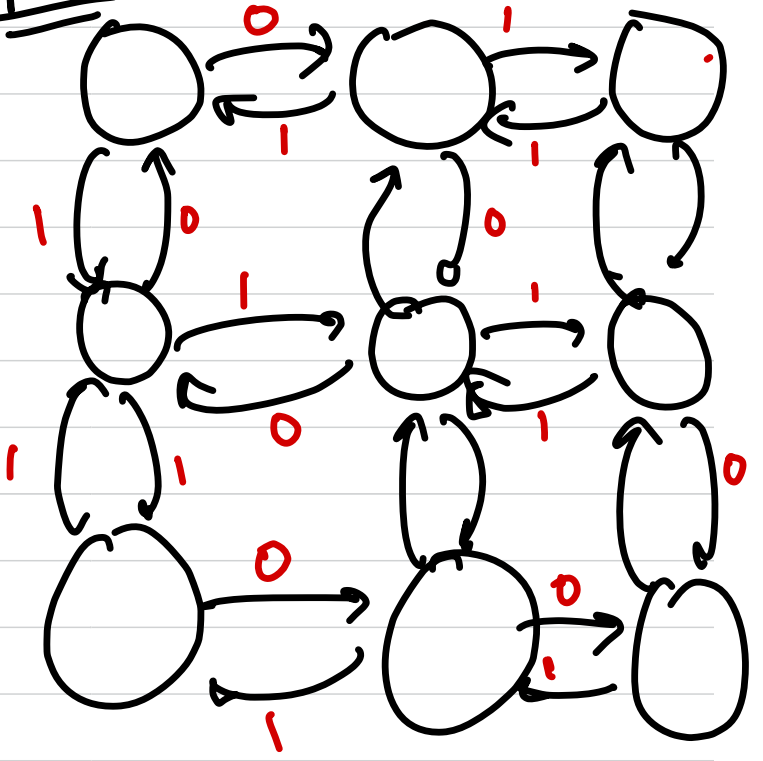
BFS

Construct a graph

n=3

Directed graph

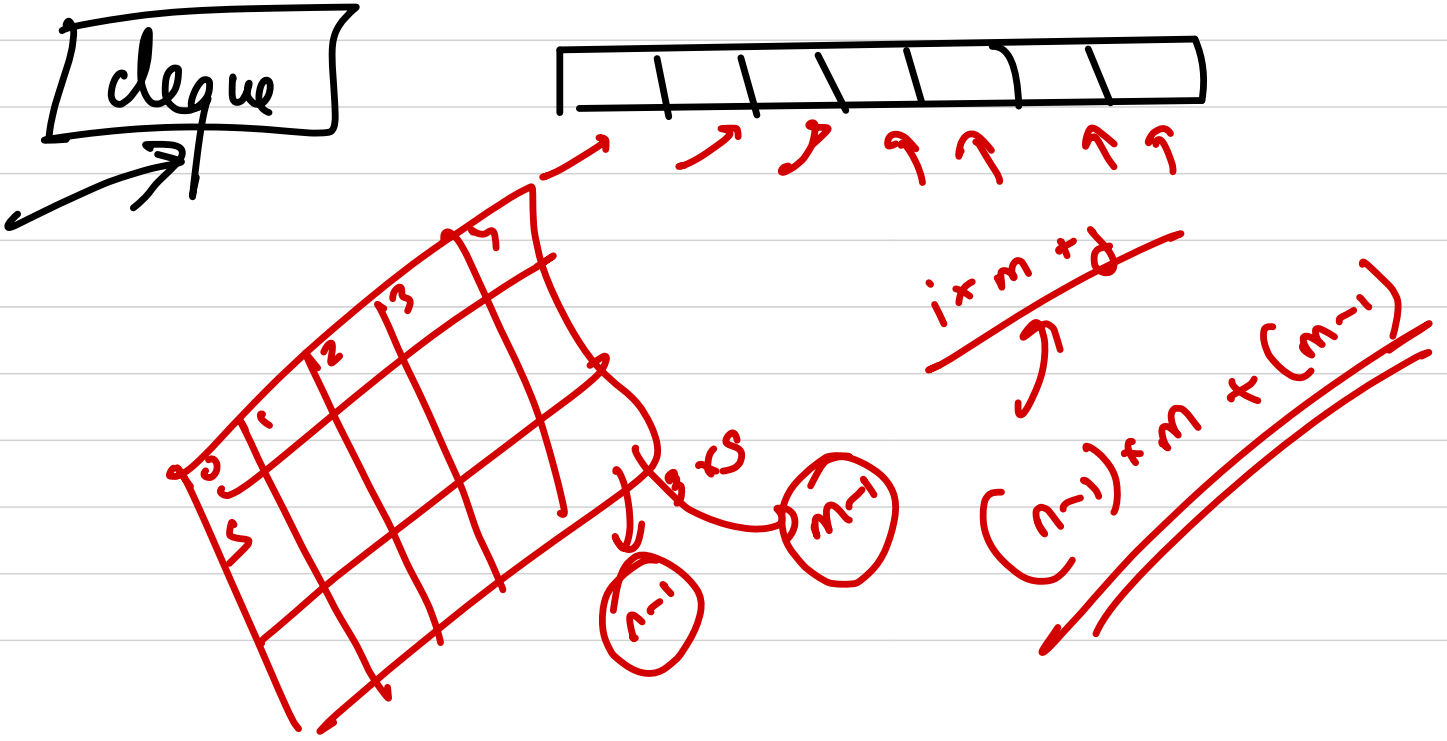


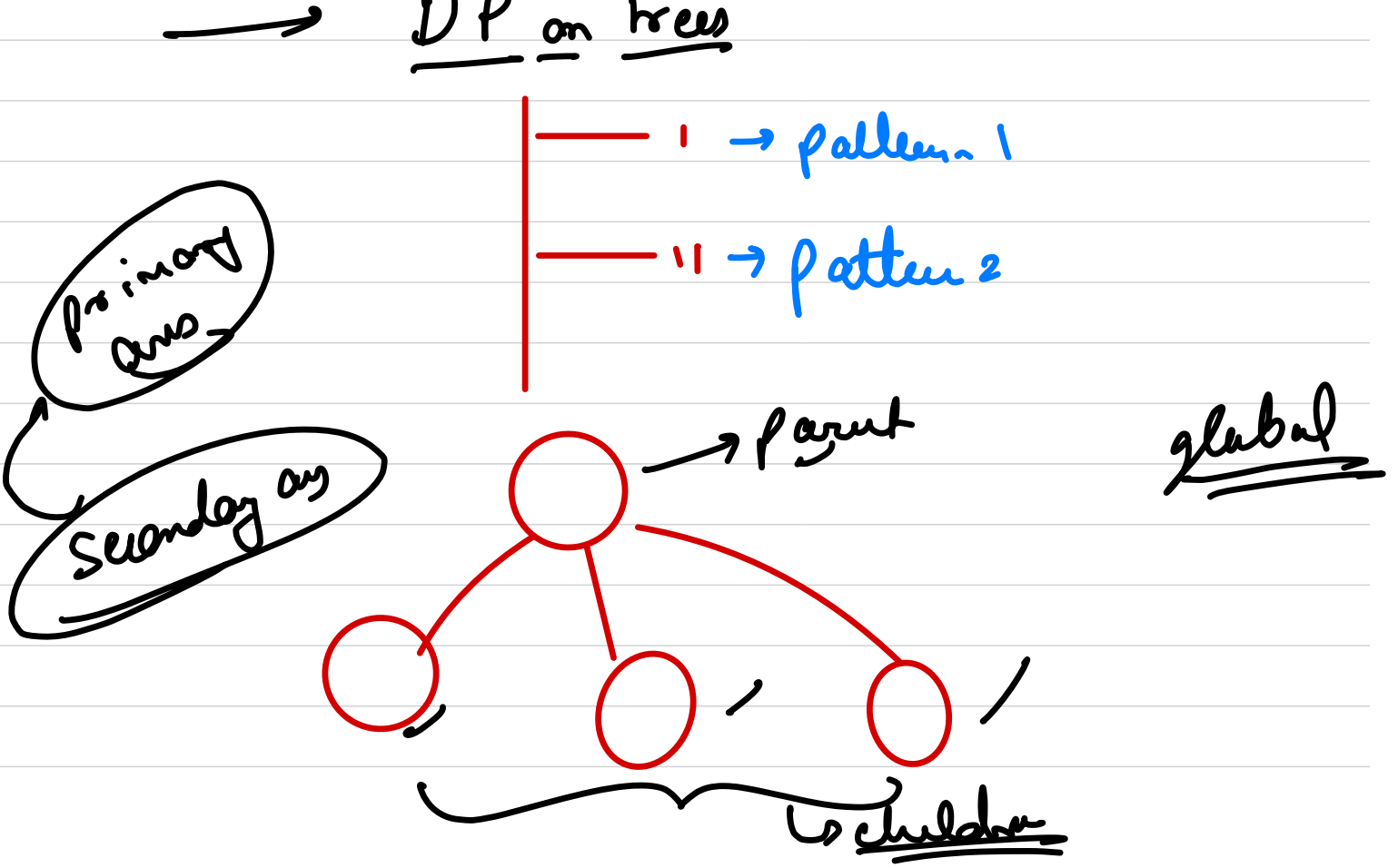


greedy

0-1 Bfs

deque




DP on trees
I → pattern 1
II → pattern 2
Primary axis
Secondary axis
parent
global
children

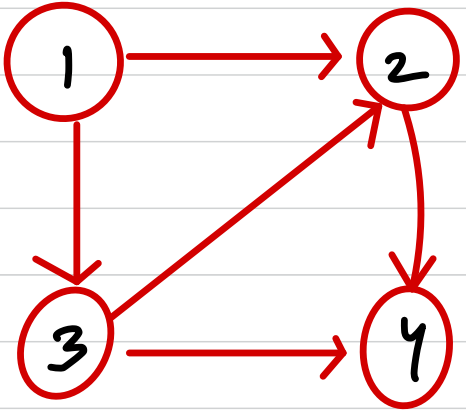
longest path → atcoder

$f(x) \rightarrow 1 + \max(f(\text{nxt}))$

longest path

$\max(f(i))$ ← ans

$\forall i \in \{\text{nodes}\}$

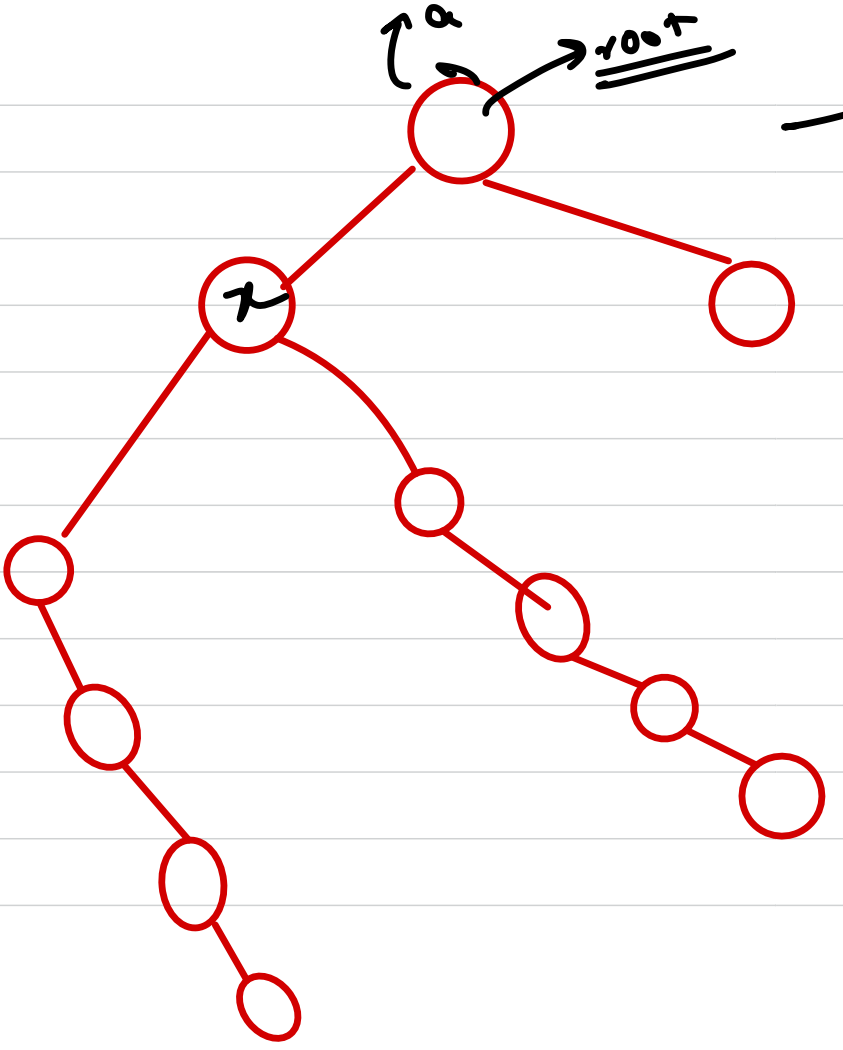


Tree

ans → 3

11

Binary Tree → Diameter



→ Binary tree

→ Diam → left

→ Diam → right

→ Diam → root

$f(v) = g(v \to left) + g(v \to right) + 2$

$f(1)$

diameter of tree passing through v

$g(x) \to$ height of x

$O(n^2)$

ans $\to$ max $(f(v), f(v \to left), f(v \to right))$

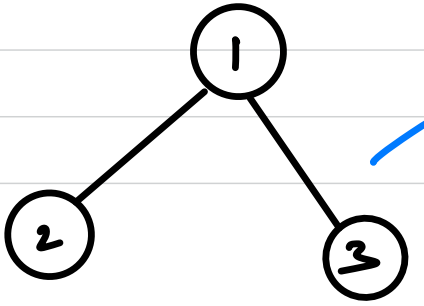
$f(u) = \{ \max(f(u \to left), f(u \to right) + 1)$

2

<height, diameter> max(left→d, right→d, left→h + right→h + 2)

Q: You own a tree with N vertices. You need to do colouring of the vertices into either blue or red. You can't color two immediate neighbours or adjacent vertices with blue. Find no. of ways to paint tree.

$n \leq 10^5$

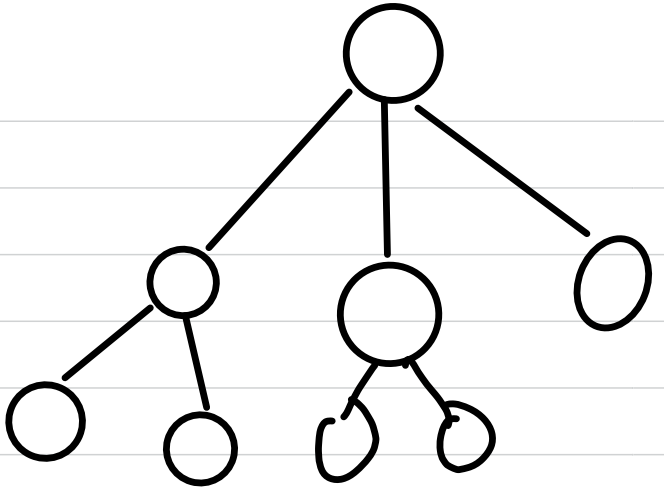


5 ways

$f(v, c)$

White → Red
Black → Blue

$c \rightarrow 0, 1$



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

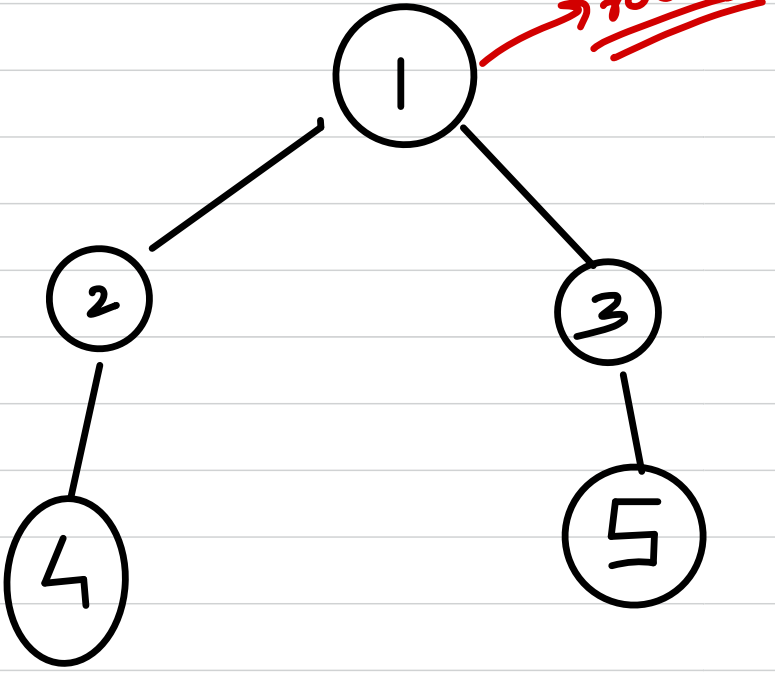
$f(v, 1)$ → # of ways to color subtree rooted at v → v can be R

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

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

find $q_{ms} \rightarrow f(root, 0)$

★ Subordinate → CSES



$f(u)$ → $\forall u$

↓

no of subordinate for u

$f(u) = \sum(1 + f(c))$ — dp

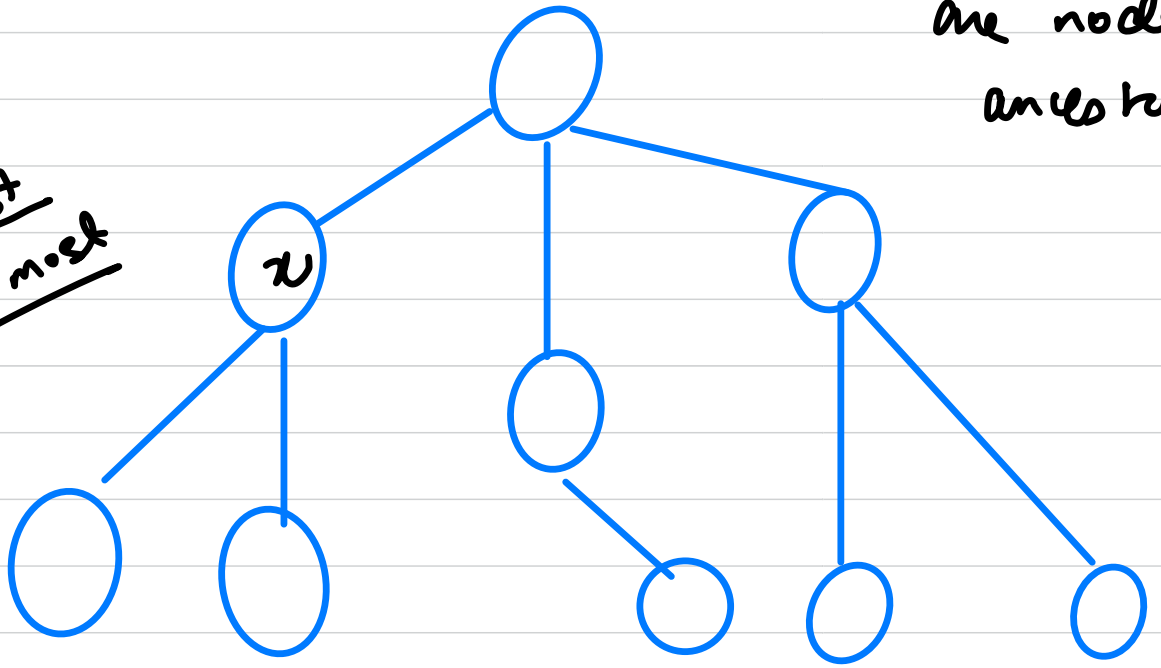
$c \in$ (children)

$\forall c$

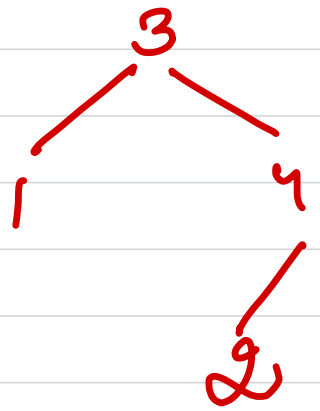
famtree → Codechef

max diff such that one node is the ancestor of other

min most
maximum most



ans = $-\infty$



(min, max)

→ secondary ans

(p, p1)

$f(u) \Rightarrow$ (min, max) for the tree rooted at u.

ans = max (ans, |min − u|, |max − u|)

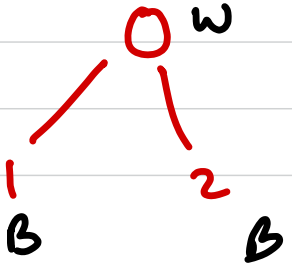
$$f(u) = \{\min(f(c)), \max(f(c))\}$$

apple man and tree → Codeforces

Divide this tree into arbitrary k components such that each component has exactly one black node.

$0 \le k < n$

Tree → (n-1)



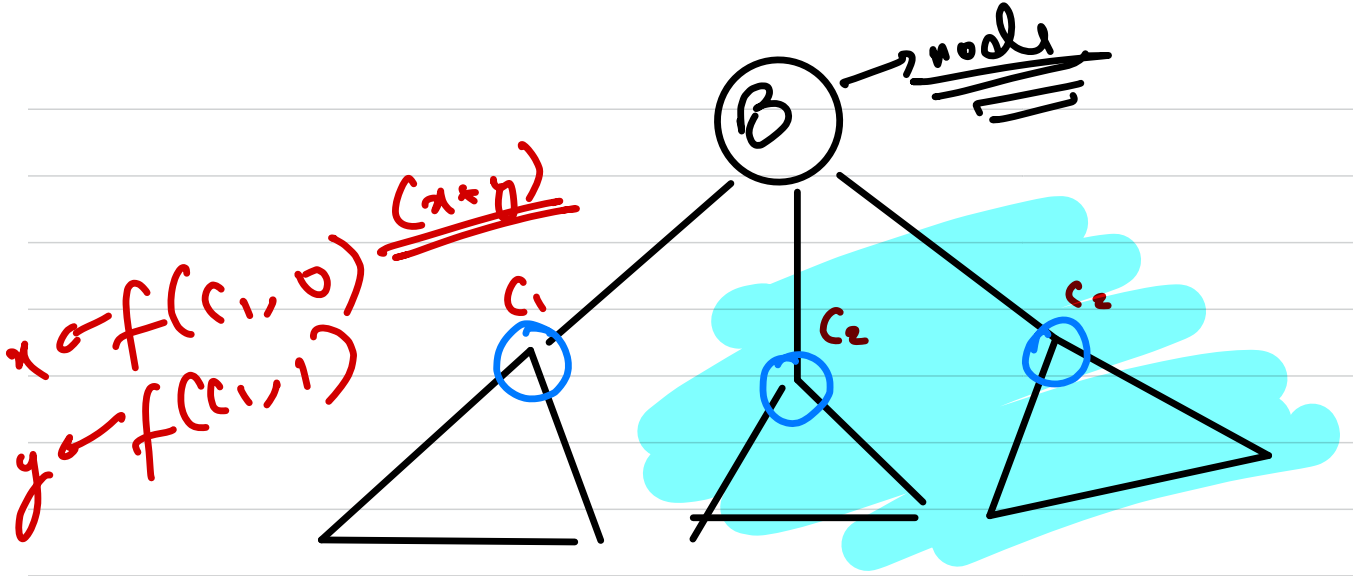
→ 2

Every node will have diffent choices based on colors of node in the subtree.

$f(node, c)$ ← count

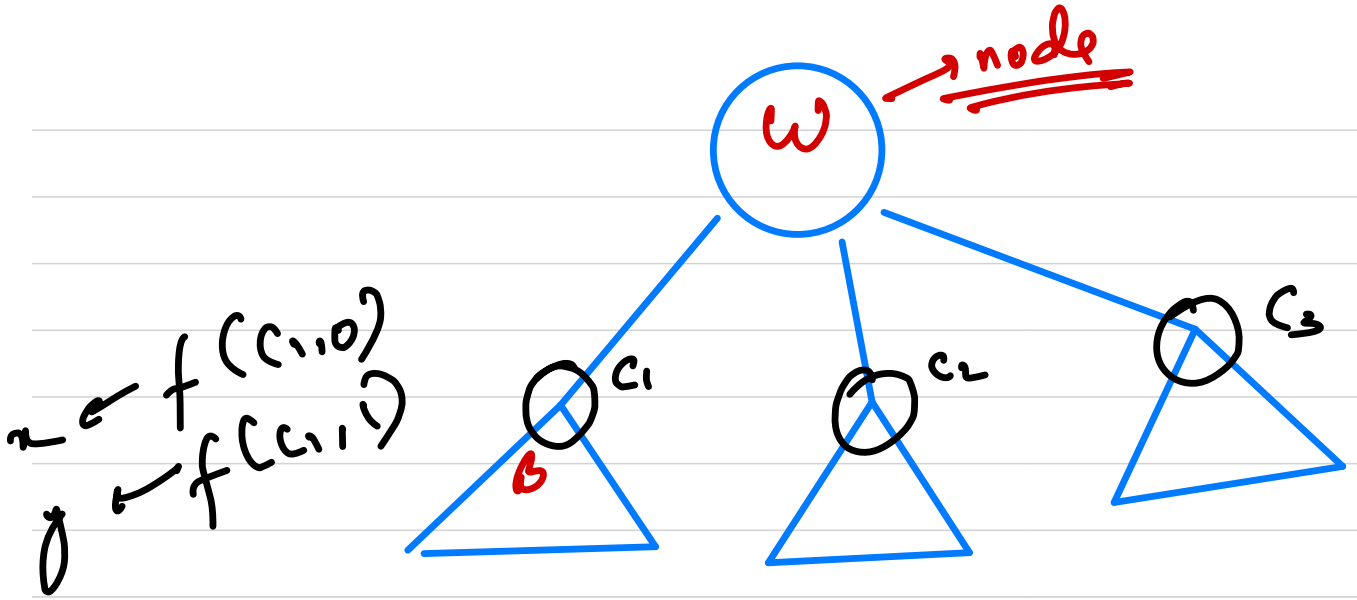
$f(node, 0)$ → # of ways subtree rooted at node has no black vertex

$f(node, 1)$ → # of ways for subtree rooted at node & has 1 black vertex



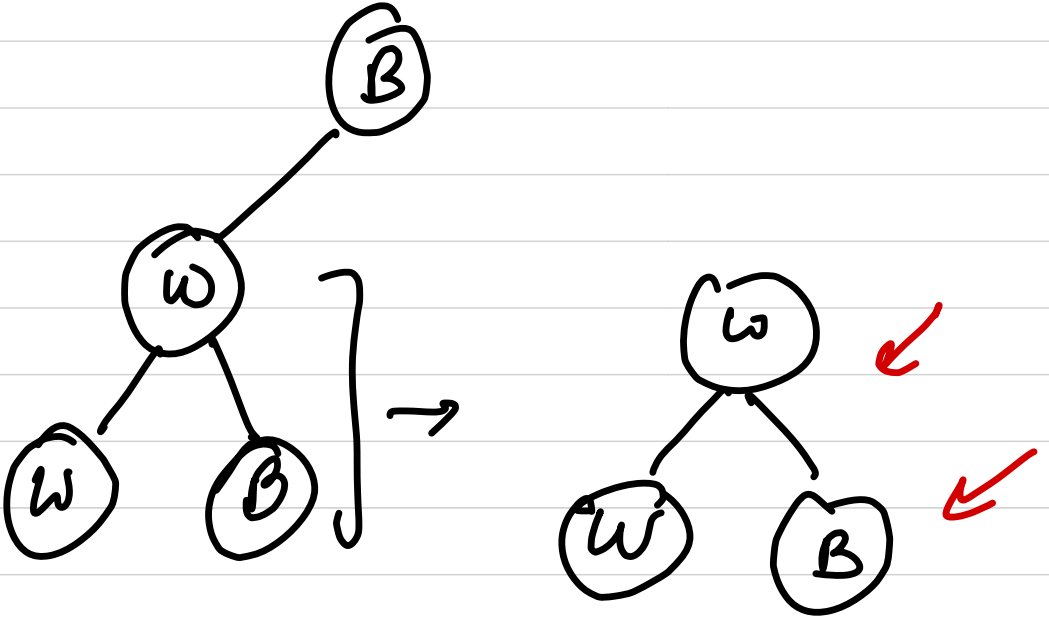
$$f(node, 1) = \prod_{c_i \in child} \left(f(c_i, 0) + f(c_i, 1) \right)$$

$$f(node, 0) = 0$$

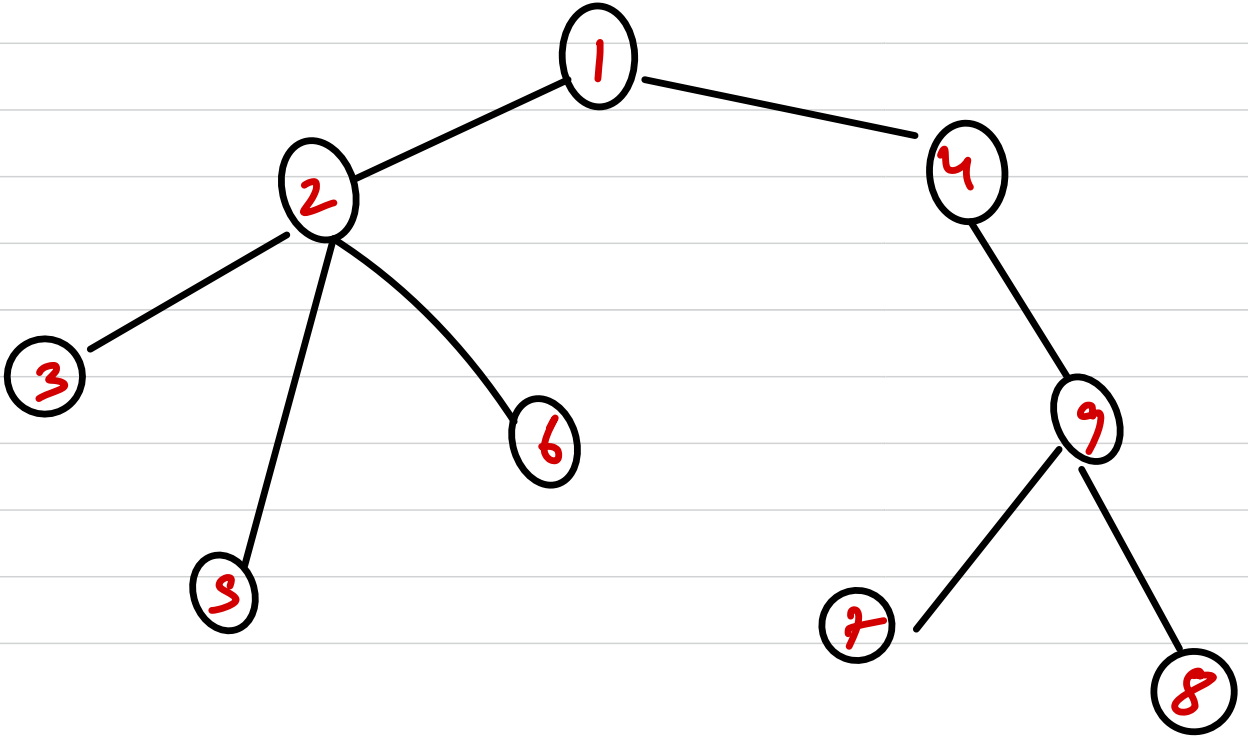


$$f(node, 0) = \pi\left(f(C_1,0) + f(C_1,1)\right)$$

$$f(node, 1) = \underbrace{\sum_{C_i \in child}} f(C_1,1) \times \left(\frac{f(node,0)}{f(C_1,0)+f(C_1,1)}\right)$$


B
W
W
B
W
W
B

Tree painting - Codeforces

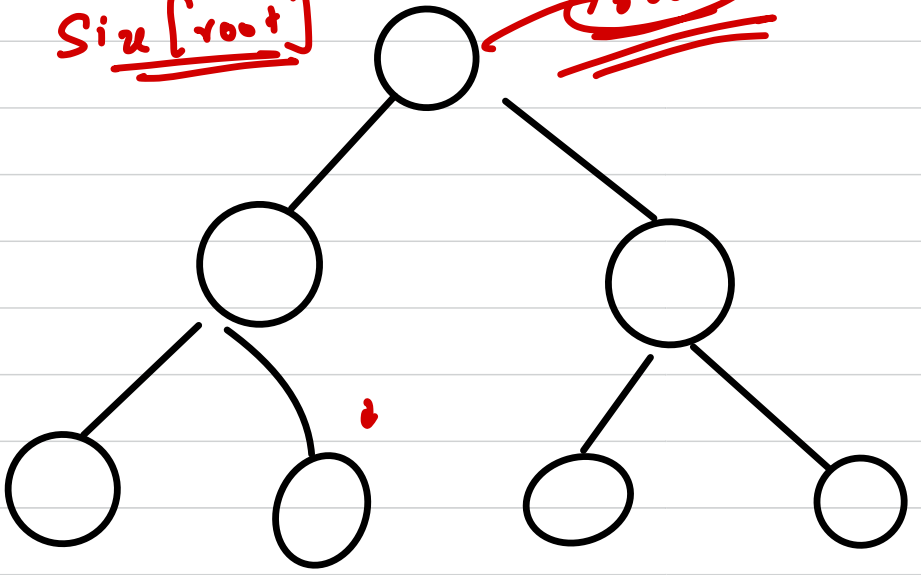


Size[root]

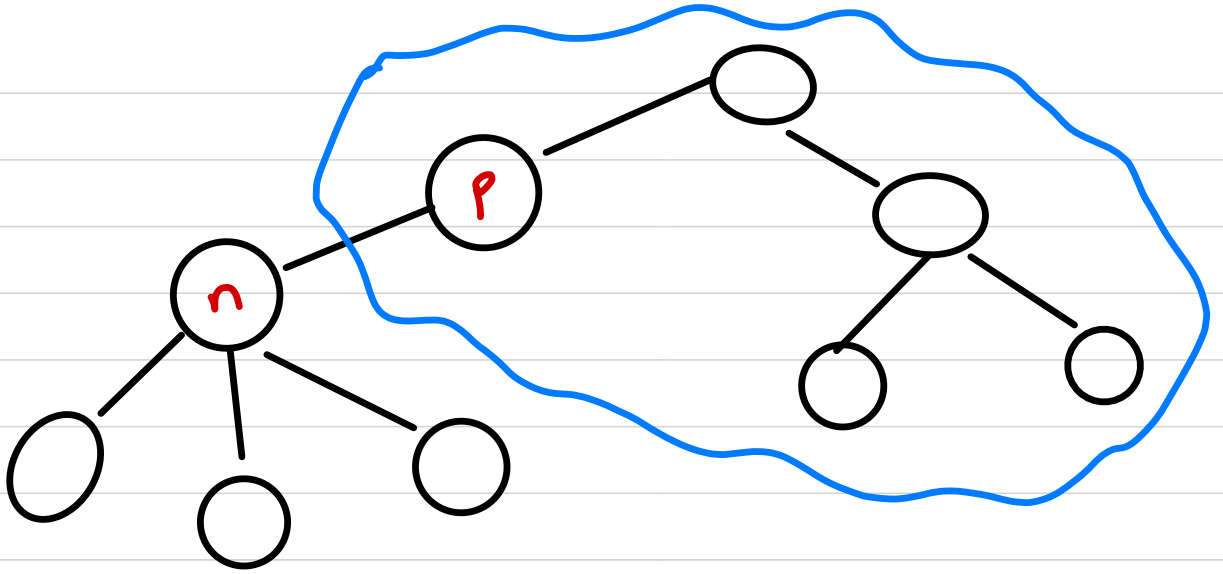
root

7+3+1+1

TLE

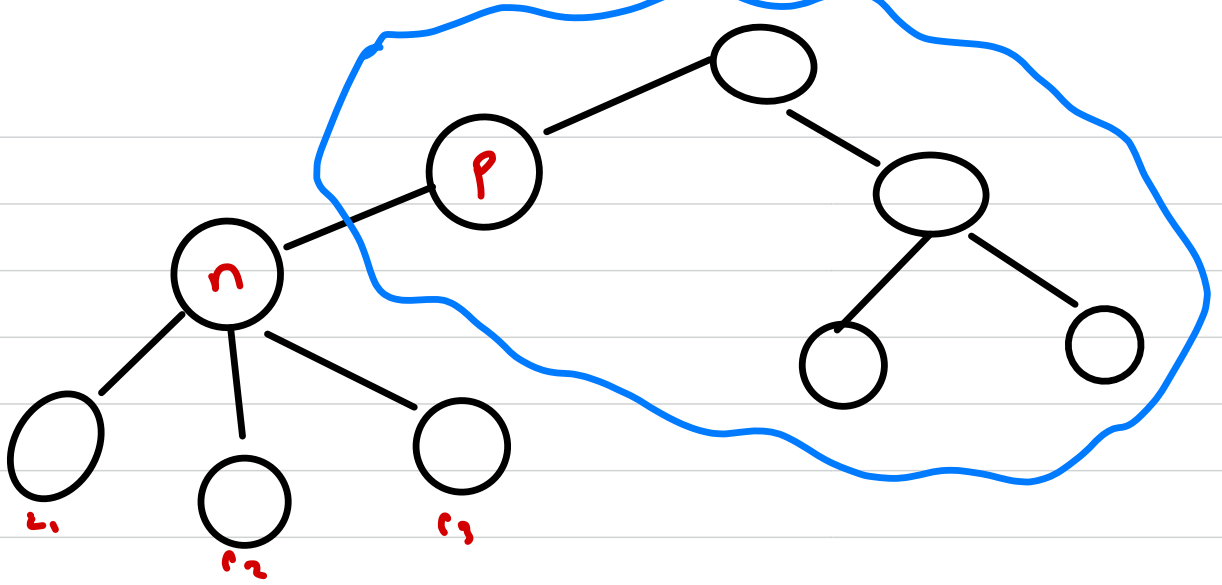


$$f(root) = \sum_{C_i \in children} f(C_i) + subtree[root] + 1$$



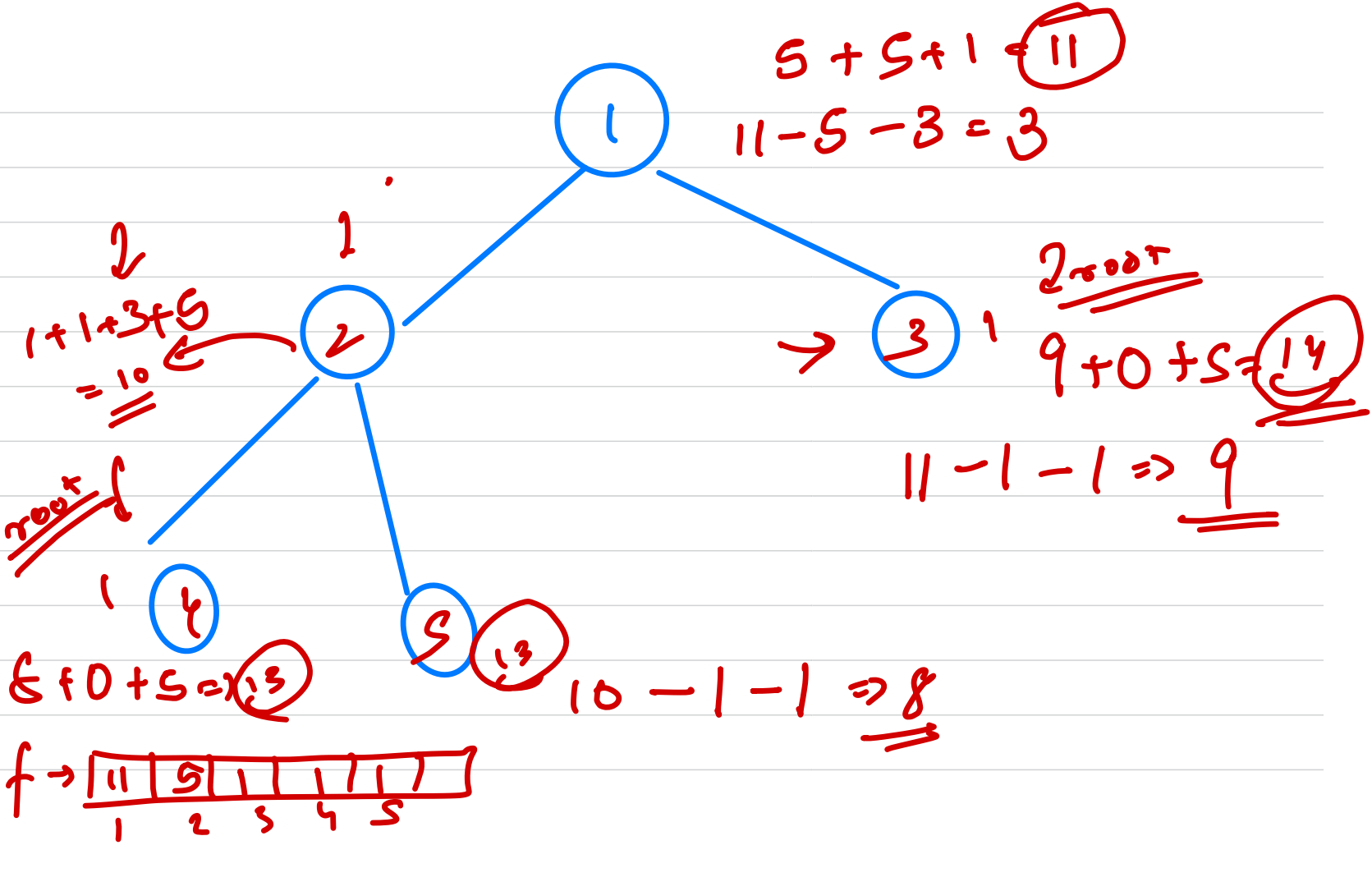
$g(p)$ → contribution of any node p as an inverted parent.

$h(p)$ → Size of this inverted parent subtree

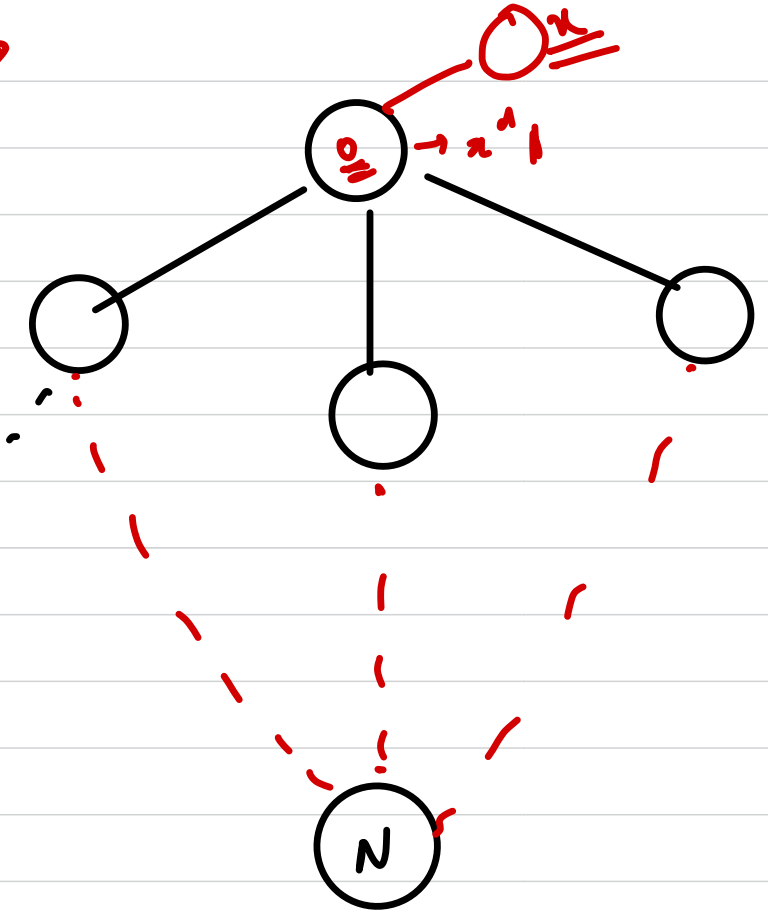


$$f(n) = \sum f(c_i) + g(p) + \text{subtree} + 1$$

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


5+5+1=11
11-5-3=3
1
2
1
1+1+3+5
=10
2
root
1
4
5
13
6+0+5=13
10-1-1⇒8
3
1
2root
9+0+5=14
11-1-1⇒9
f→
11
5
1
2
3
4
5

Xor circuit →



$$f(n, x) = \sum_{i=0}^{3} \left(\pi \left(f(c_i, x \hat{} i) \right) \right)$$

node = N → 0

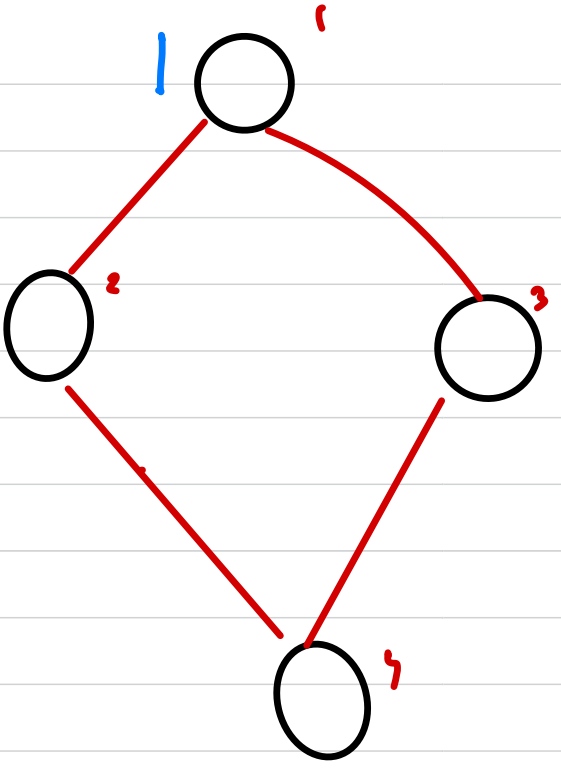
dl(n, u)

x ← (0, 2)

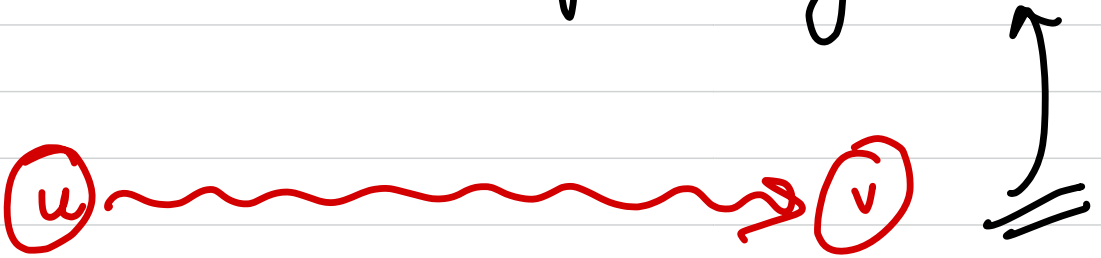
1^2=3

1^3=2

2^3=1



→ Shortest path algorithms ←



BFS → unweighted / uniformly weighted

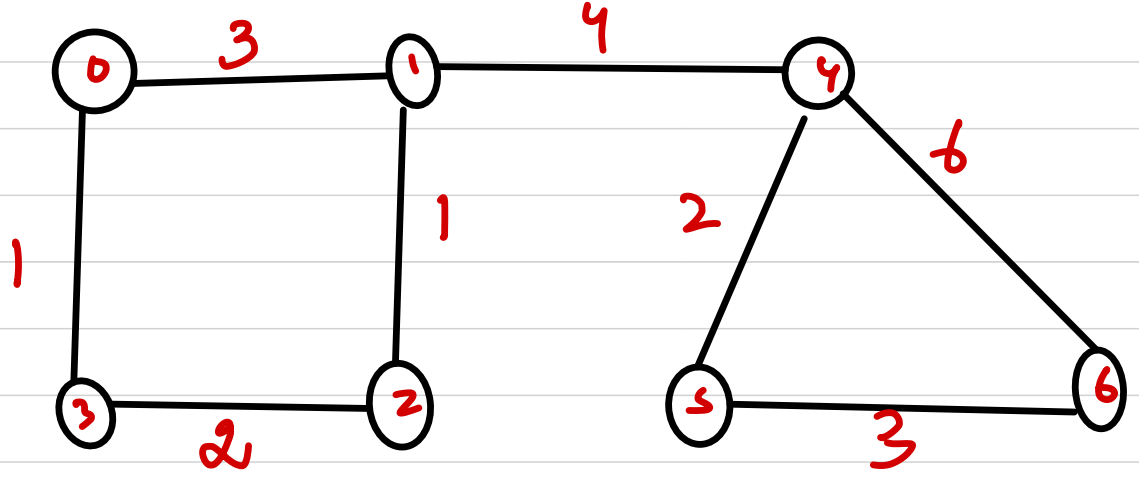
weighted → ① Dijkstra

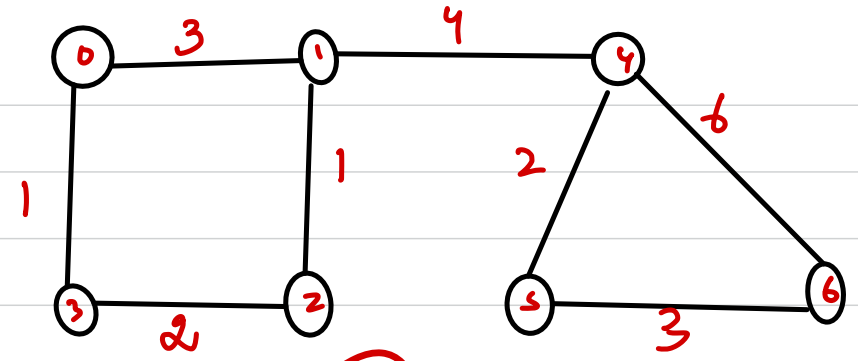
Single source shortest path

modified BFS

→ greedy algo + BFS → Dijkstra's

Src → 0





vig

Src → 0

Visited → Reached SP

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

MH

node	SD
0	0
1	3
2	3
3	1
4	7
5	9
6	12

↳ current SP from src to the node

pq

pq

if (dist[src-node] > dist[src-par] + dist[par-node])

dist[src-node] = dist[src-par] + dist[par-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 → $D(s,u) = y(s,u)$

when u is marked visited 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 :

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

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

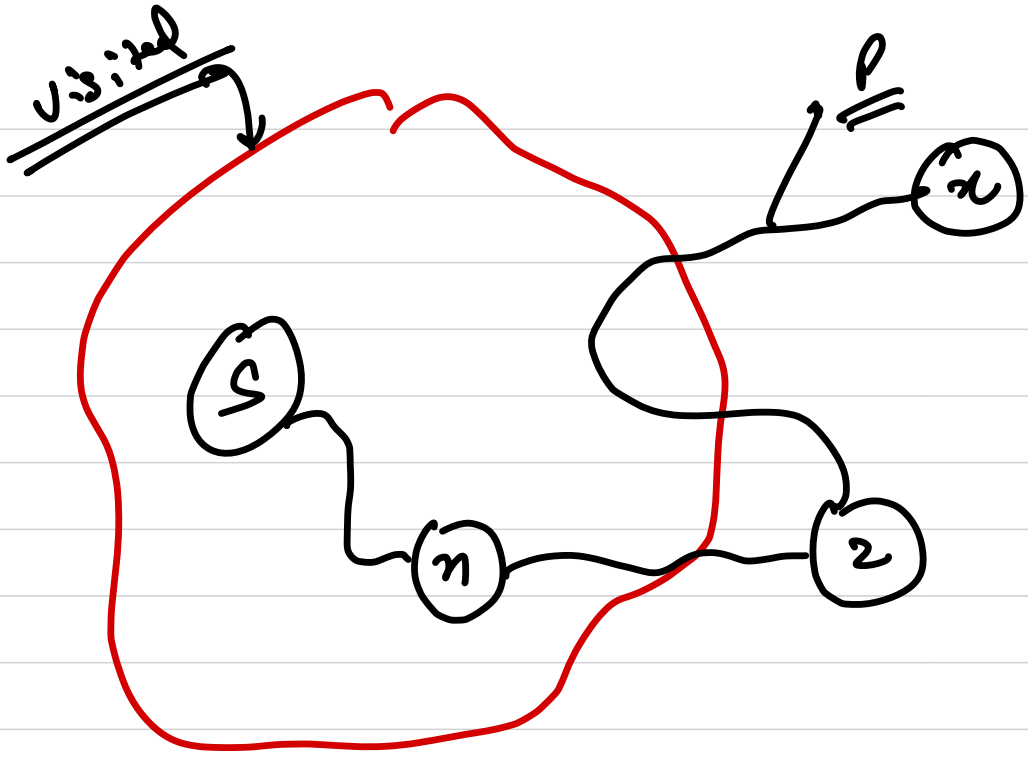
$$D(s, m) = y(s, m)$$

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 x is the predecessor of z included in the path P. and included in visited.



We say

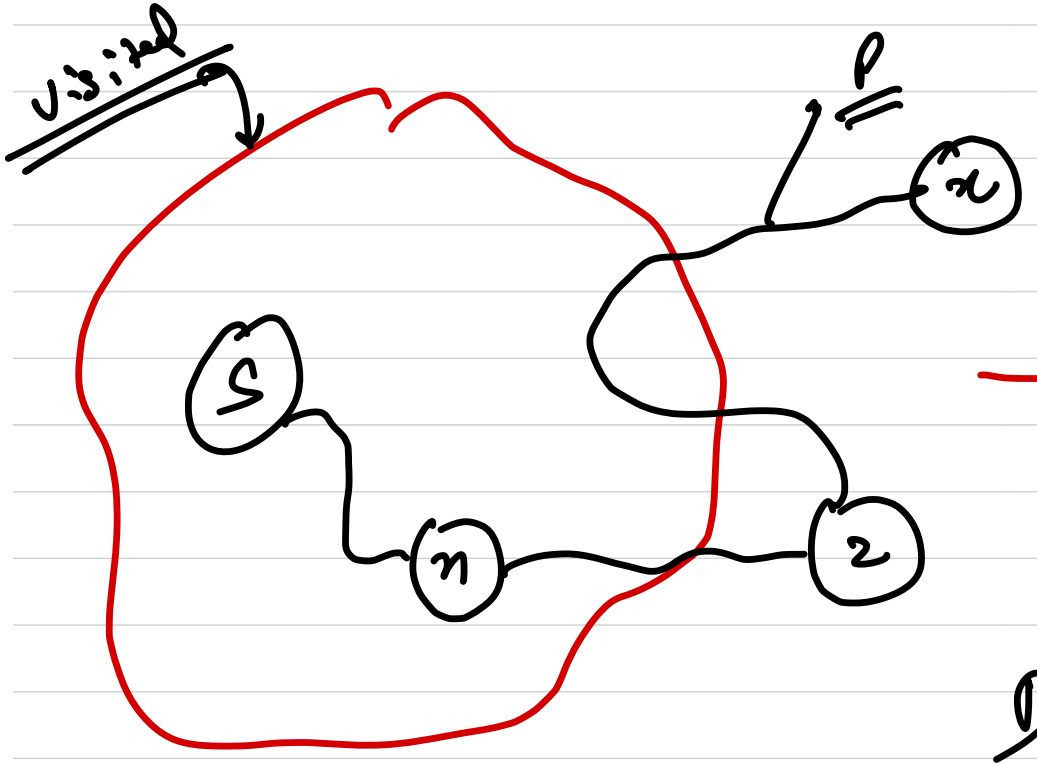
$D(s,n) = y(s,n)$ — (1)

$D(s,z) = D(s,n) + \text{edgecost}(n,z)$ — (2)

(by dijkstra)

$\hookrightarrow = y(s,n) + \text{edgecost}(n,z)$

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



$y(s,x) =$

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

$D(s,x)$

$D(s,z)$

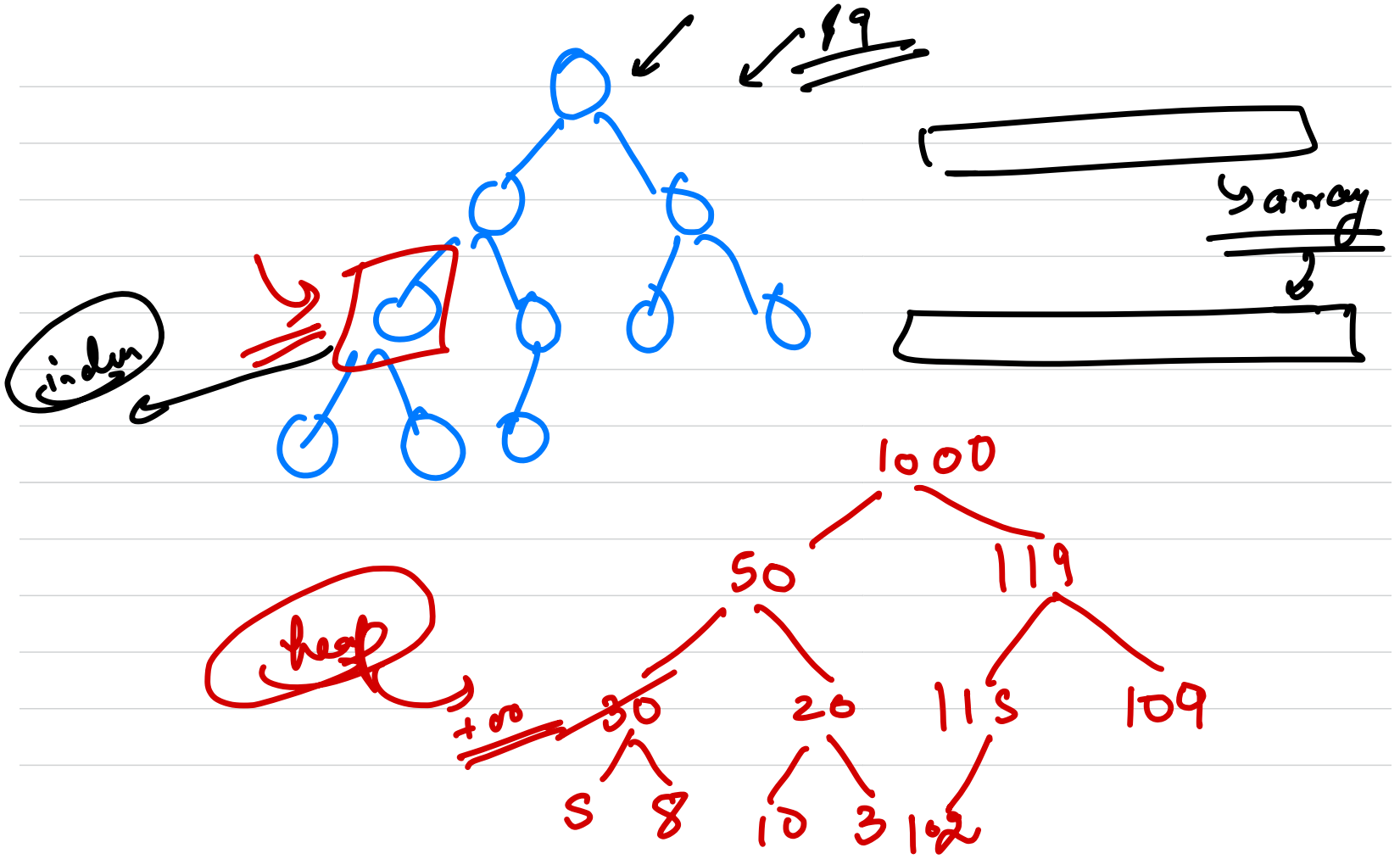
$$D(s,x) \leq D(s,z)$$

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

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

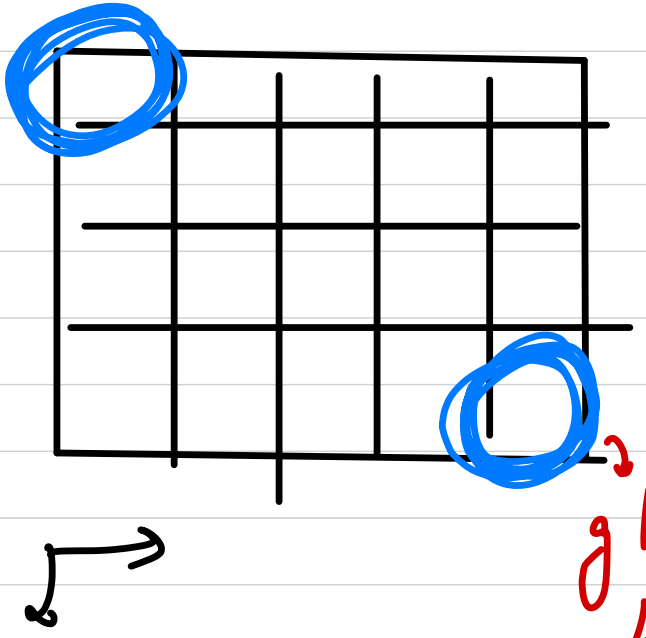
$$D(s,x) \leq \underbrace{y(s,n) + cost(n,z) + y(z,x)}$$

$$D(s,x) \leq y(s,x)$$


19
index
array
1000
50
119
heap
+∞
30
20
115
109
5
8
10
3
102

Q2

Dijkstra



grid of size

x y

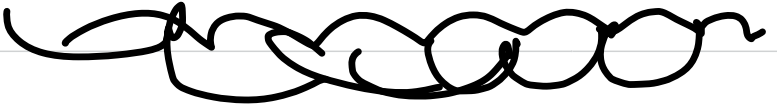
cell

g[i][j] → cost of travelling through a cell i,j

grid → no -ve weight cycle

i, j →

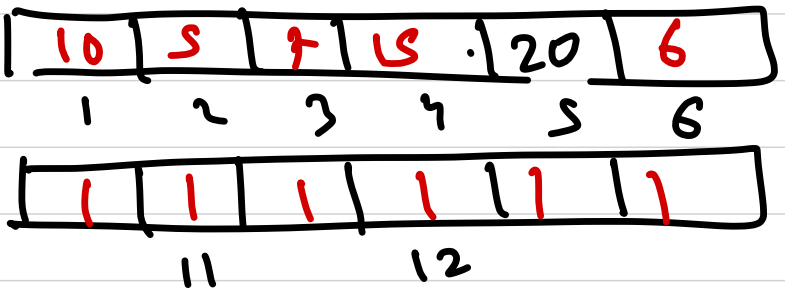
$$f(n) = f(n-1) + f(n+1)$$

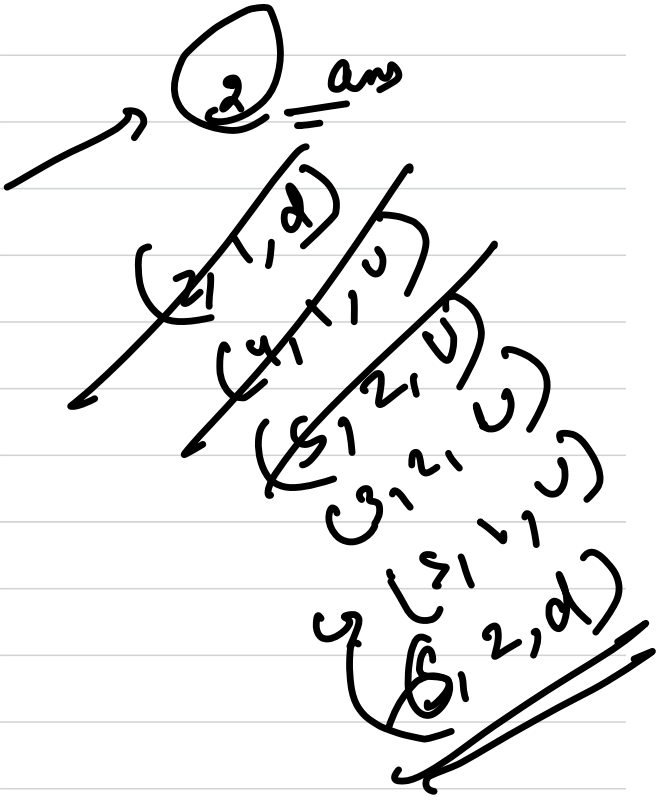


$$f(i,j) = f(i,j-1) \quad f(i-1,j)$$
$$f(i+1,j) \quad f(i,j+1)$$

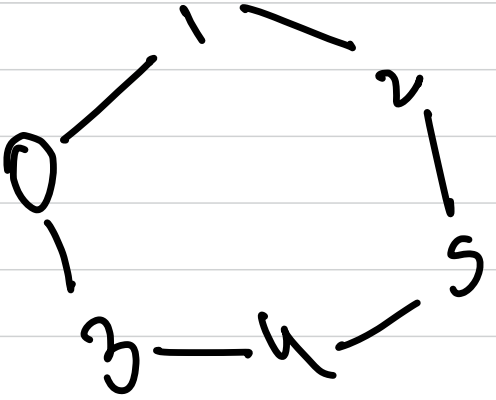
Q3 N cities are there, and i^{th} city is at hi height from base sea level. You have a few truck that requires no fuel but can go only in either "up" or "down". These cities are connected by K bidirectional roads. You want to travel from City 1 to N with min cost because, when you change mode of the truck you need to bear Ci cost if you are at i^{th} city currently. if impossible print -1

N → 6 K = 6





1 - 1
2 - ∞
3 - ∞
4 - ∞
5 - ∞
6 - ∞

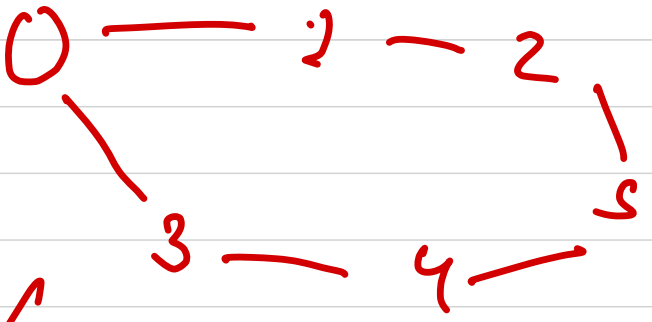


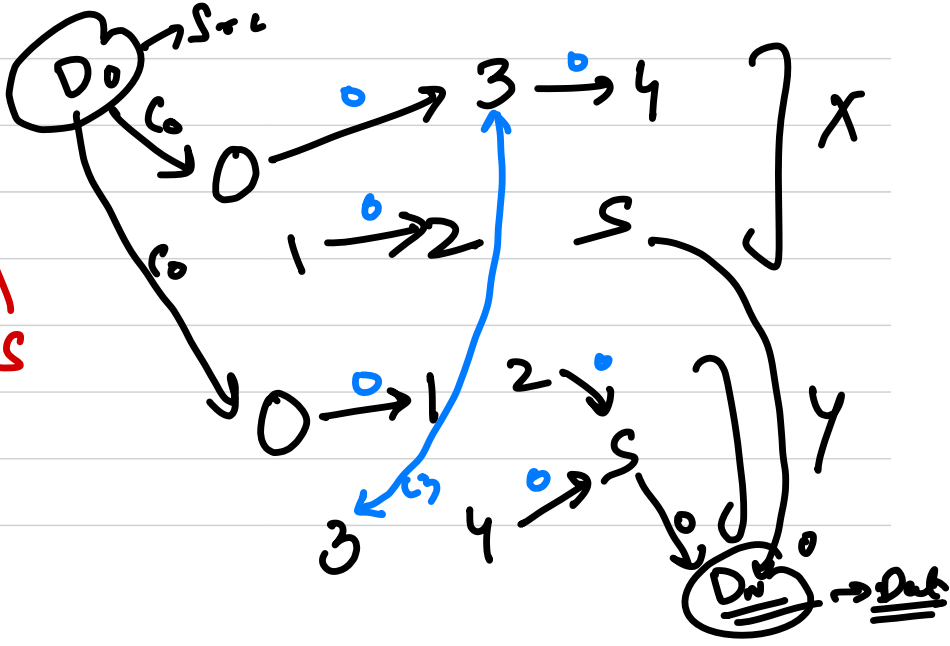
10 5 7 15 20 6

1 1 1 1 1 1

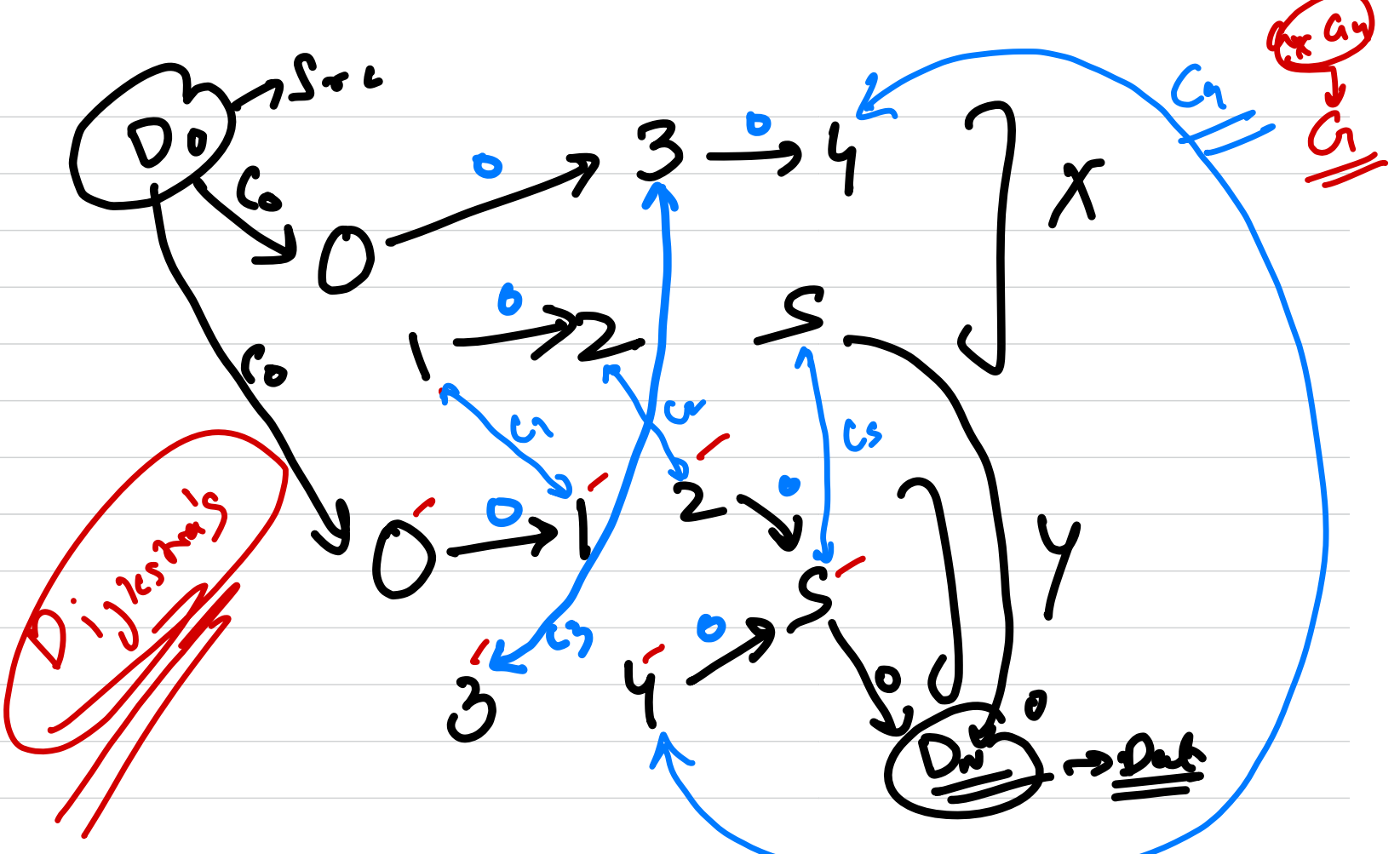
0 1 2 3 4 5

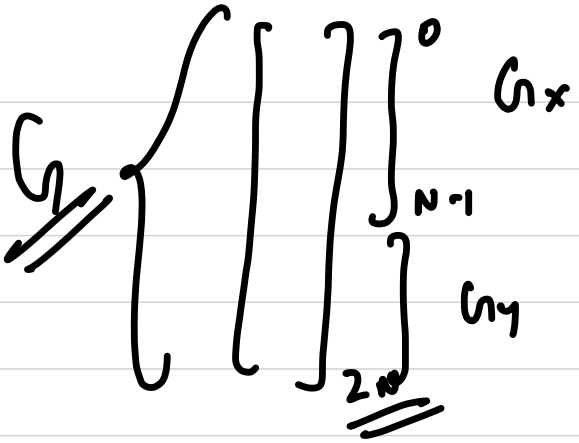
$h_1 \geq h_2$





CWS

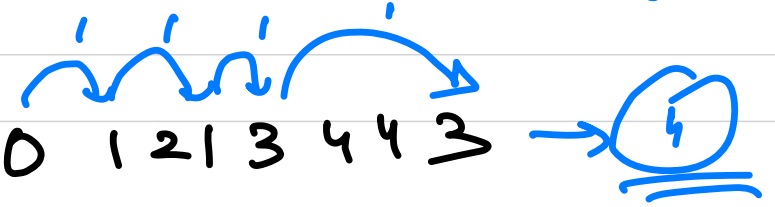

Dijkstra's
Src
Dst
x
y
0
1
2
3
4
5
0'
1'
2'
3'
4'
5'


0
N-1
2N

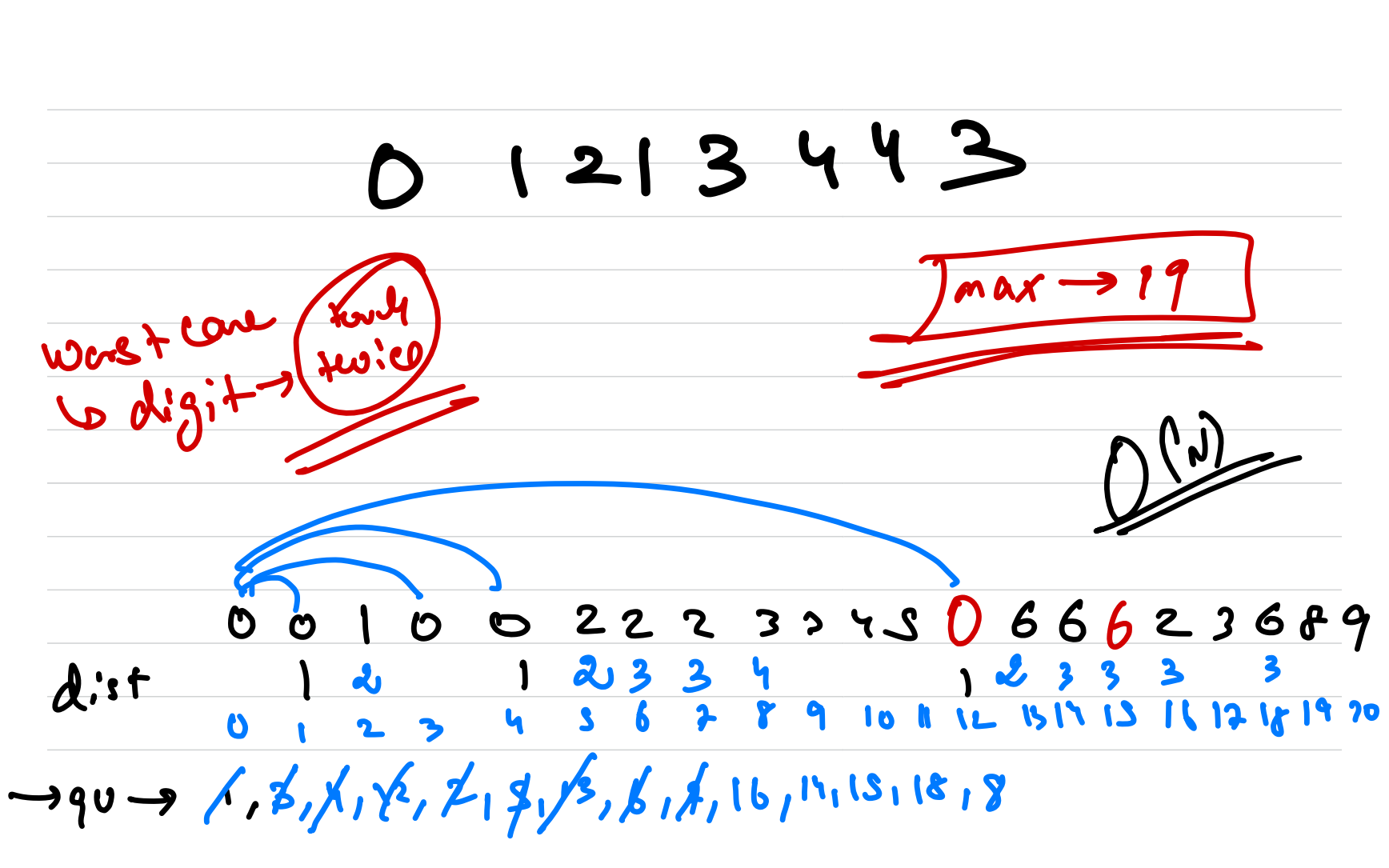
Q. Given a string of digits of length N. ($N \leq 10^5$). You're standing at 0^{th} index & want to reach at $(N-1)^{th}$ index.

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.



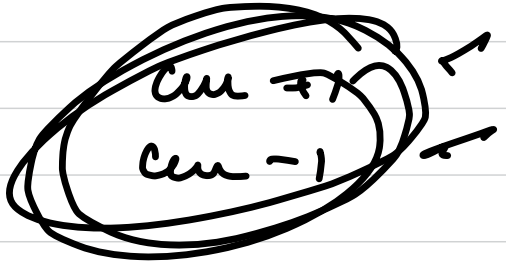
[0,9] → digits


0 1 2 1 3 4 4 3
worst case to digit → touch twice
max → 19
O(N)
0 0 1 0 0 2 2 2 3 4 5 0 6 6 6 2 3 6 8 9
dist
1 2 1 2 3 3 4 1 2 3 3 3 3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
→ qu →

digit → occ → i, j, k, l

0 → digit → BFS

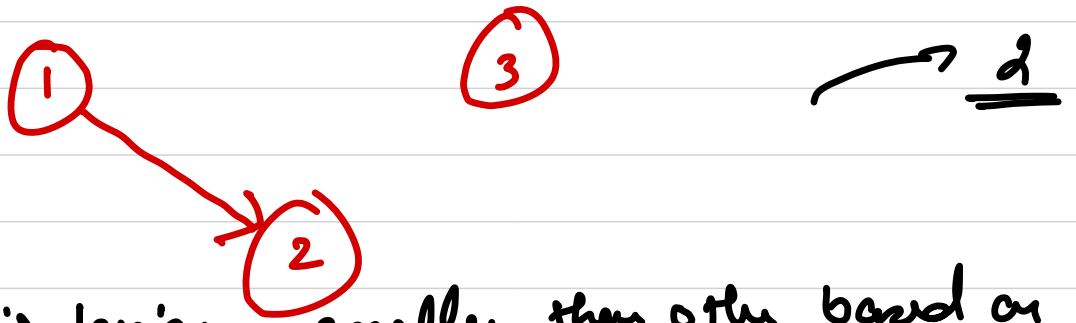
cur ← q.pop



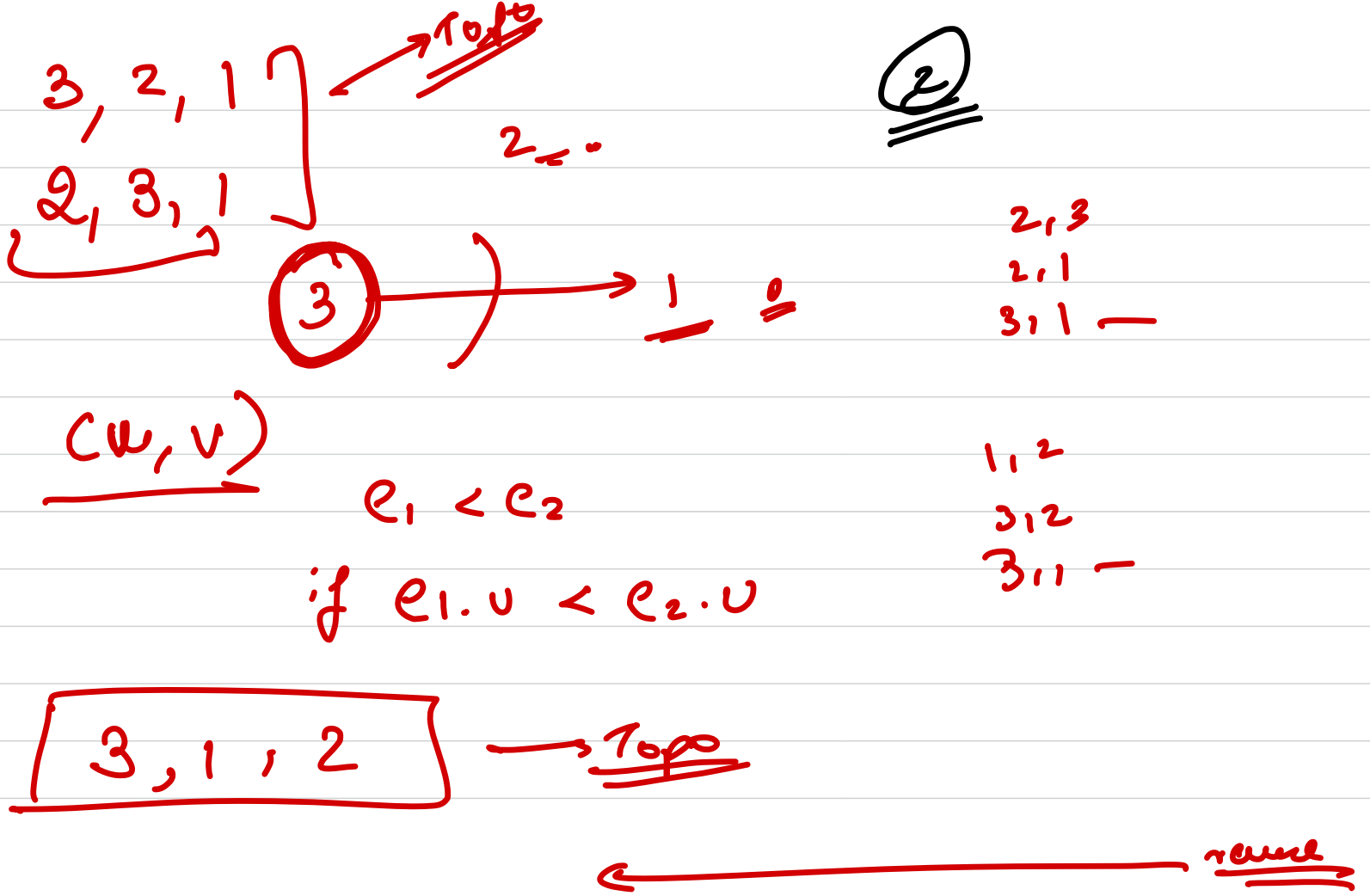
- push

Q2 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. Print in lexicographic order

$N \leq 1500$

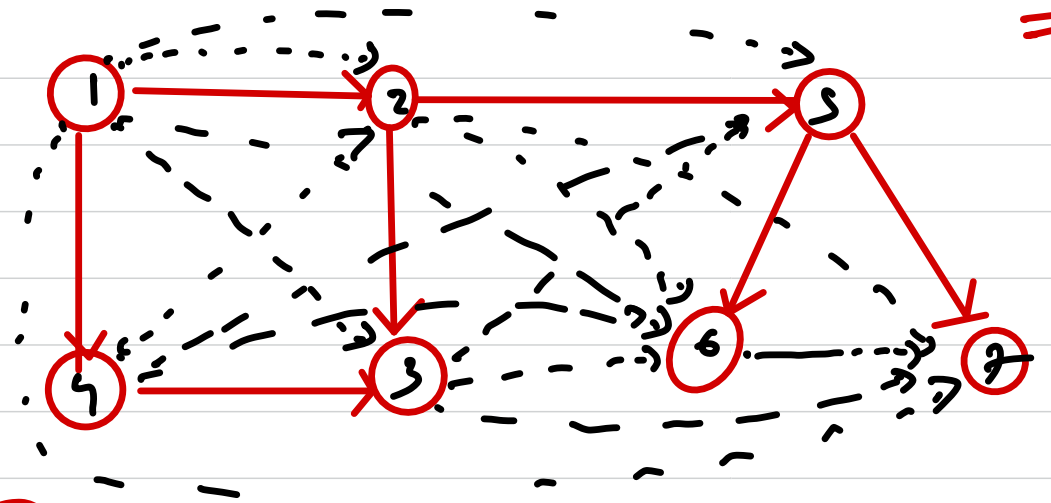


A sequence of edge is lexico-smaller than other based on the first edge differing in the given sequence


3, 2, 1
Top
2, 3, 1
3
1
2
(2)
2, 3
2, 1
3, 1 —
(u, v)
e1 < e2
if e1.v < e2.v
1, 2
3, 2
3, 1 —
3, 1, 2
Top
reverse

$u \to v$ → given edge

$v \to v$ $x < v$



→ 1, 4, 2, 3, 5, 6, 7 → Topo order

(u, v) $u < v$ then we can add an edge from $u \to v$ if it doesn't exist

$E \leftarrow$ maximal set

E' → Topo $|E'| \geq |E|$

outdegree → 0 → (max magnitude) → prio

$(u < v)$

$U_0 = 0$

$V_0 = 0$

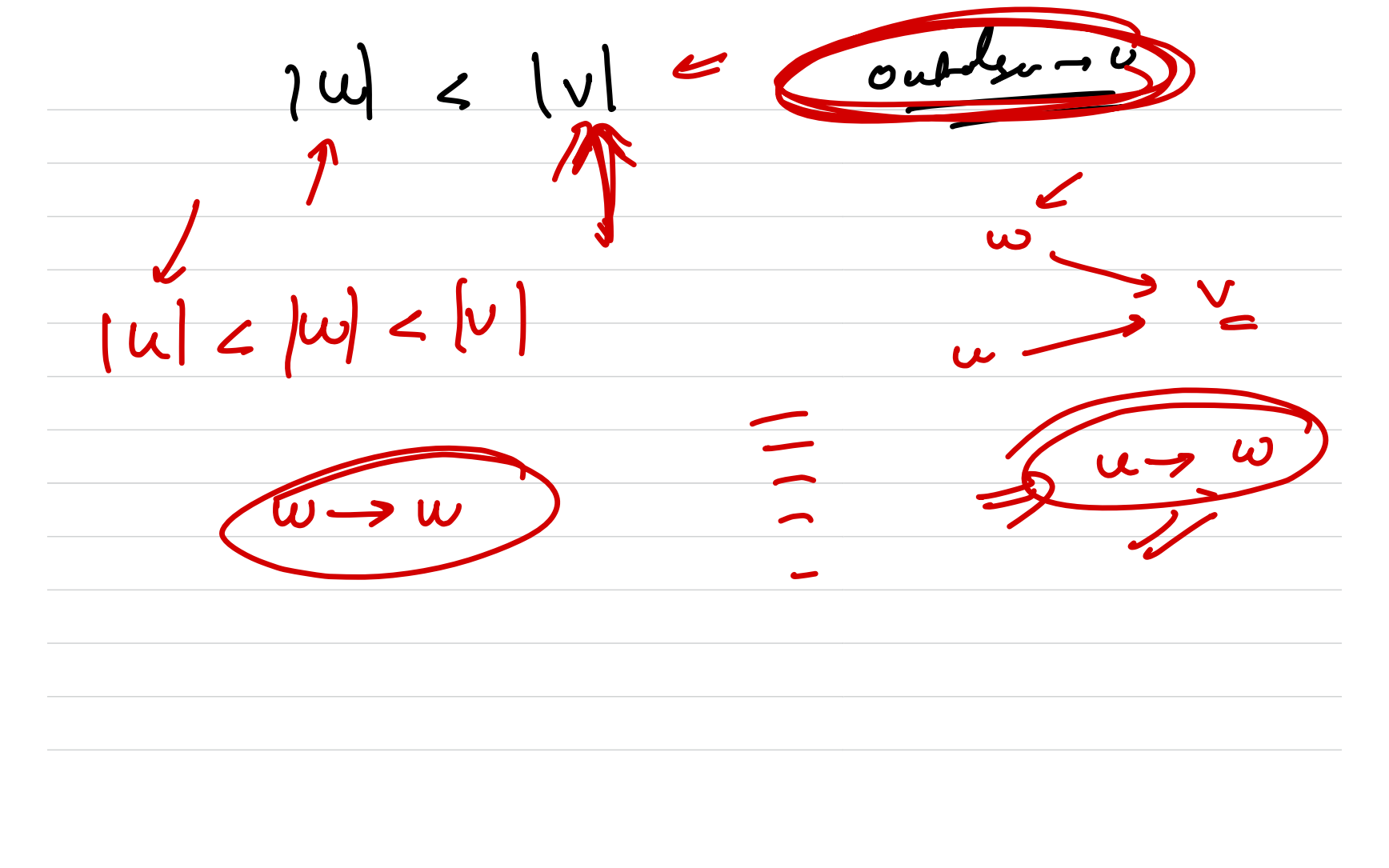
$u < w < v$

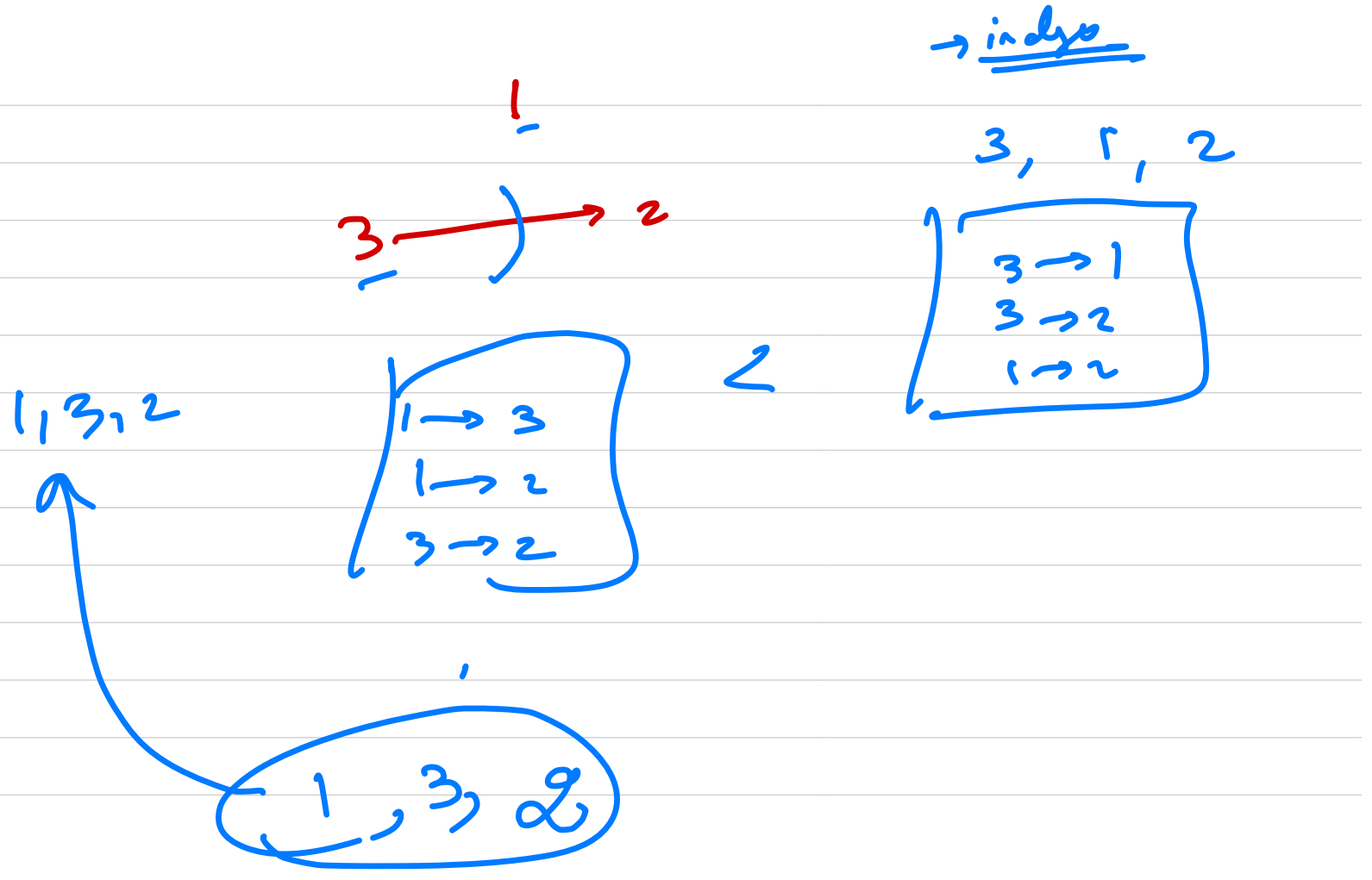
$u \rightarrow v$

$w \rightarrow v$

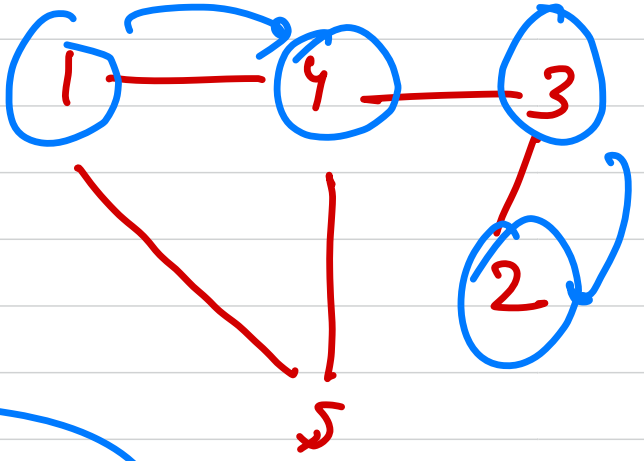
$w \rightarrow u$

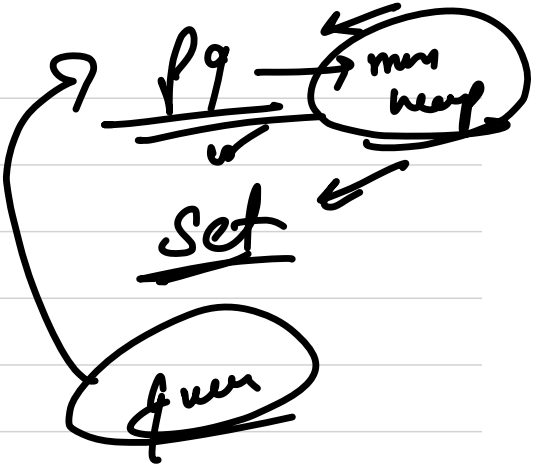
$v \rightarrow w$


|w| < |v|
outputs w
|u| < |w| < |v|
w
u
v
w → w
u → w


→ indyo
1
3 → 2
3, 1, 2
3 → 1
3 → 2
1 → 2
1 → 3
1 → 2
3 → 2
1, 3, 2
1, 3, 2

$O(n+m)$





1, 4, 3, 2, 5

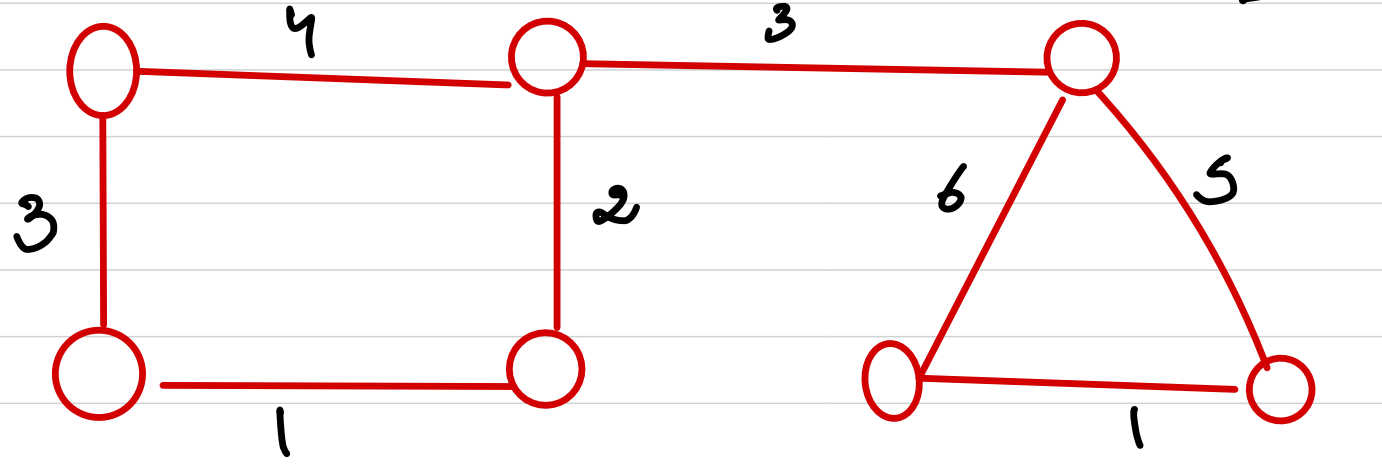
Minimum Spanning Trees

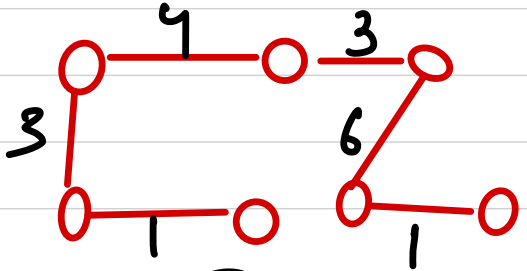
Tree → graph without cycles

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

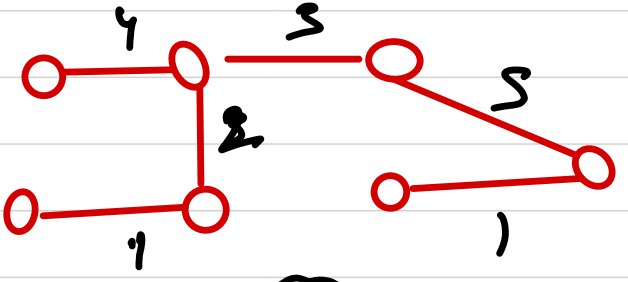
MST → Spanning tree with minimum total edge weight.

G



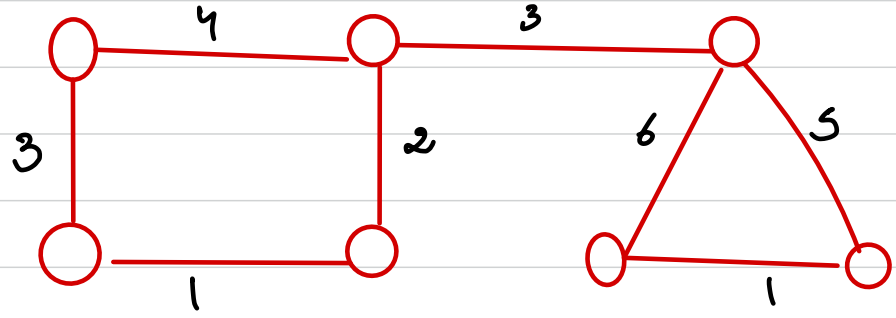


T8



16

greedy ← Kruskal's Algorithm]



avoid cycles

cycle detection

DFS

DSU

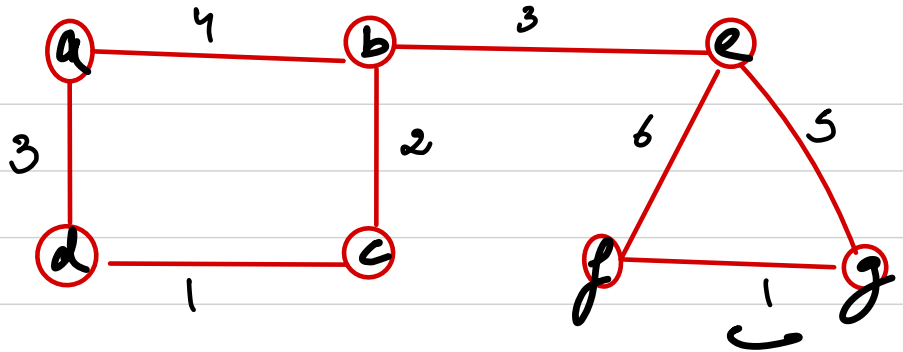
Standard

edges pick → greedy from min wt edge such that it doesn't form cycle.

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

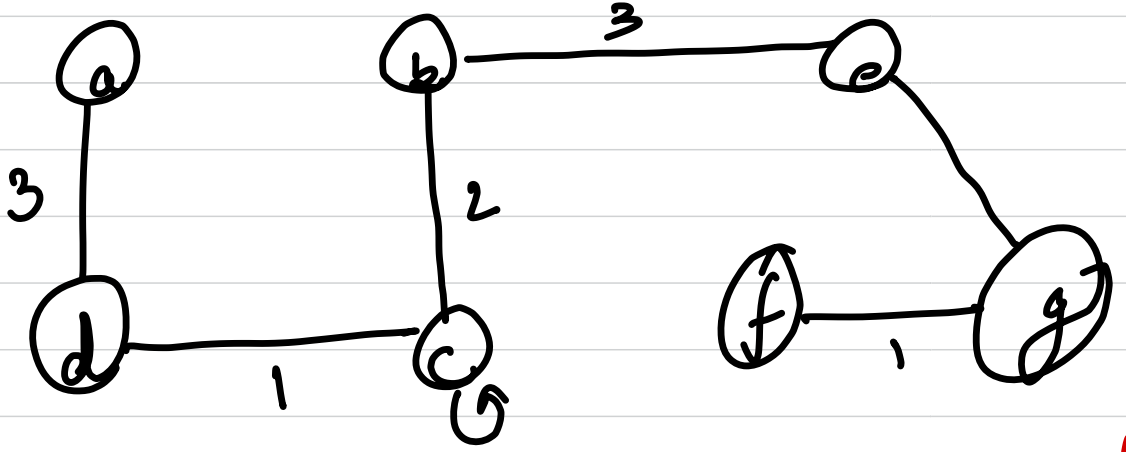
1k → un

$$\Sigma' = (\, e_1' \quad e_2' \quad e_3' \quad \cdots\cdots \quad e_{n-1}' \,)$$



← G

edges.sort()



← T

$E \log E$ ↑

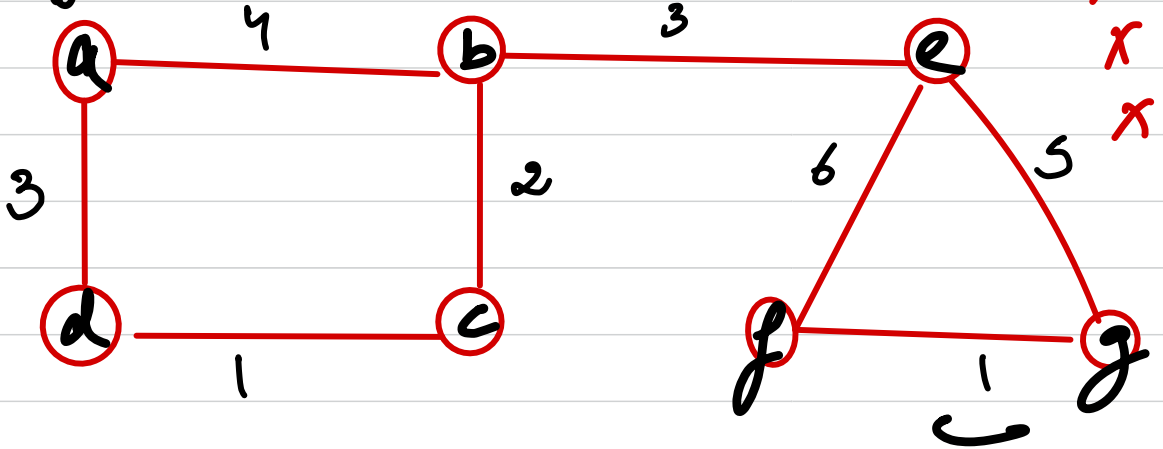
$E \log^* V$

$O(E \log E)$

greedy

Prims

src



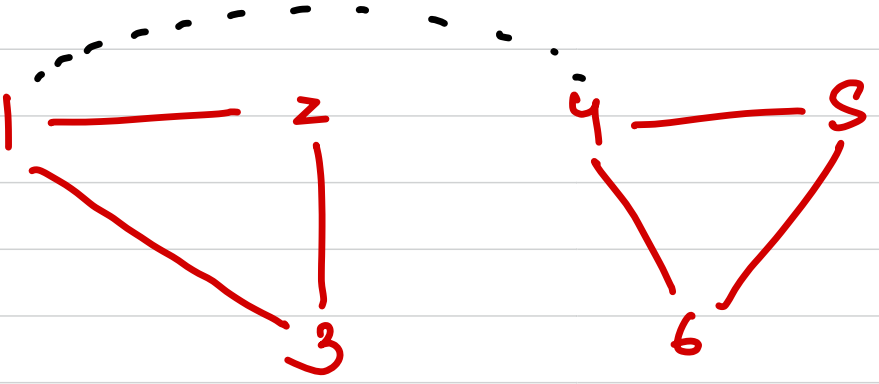
		cw	P
x	a	0	-1
x	b	2	c
x	c	1	d
x	d	3	a
x	e	3	b
x	f	1	g
x	g	5	c

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

Q. 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 btw nodes u & v if $cost_u \geq 0$ && $cost_v \geq 0$.

Construction of edge costs → $cost_u + cost_v$.

find min cost to connect the graph.



node cost

1 — 1

2 — 3

3 — 5

4 — 2

5 — 4

6 — 6

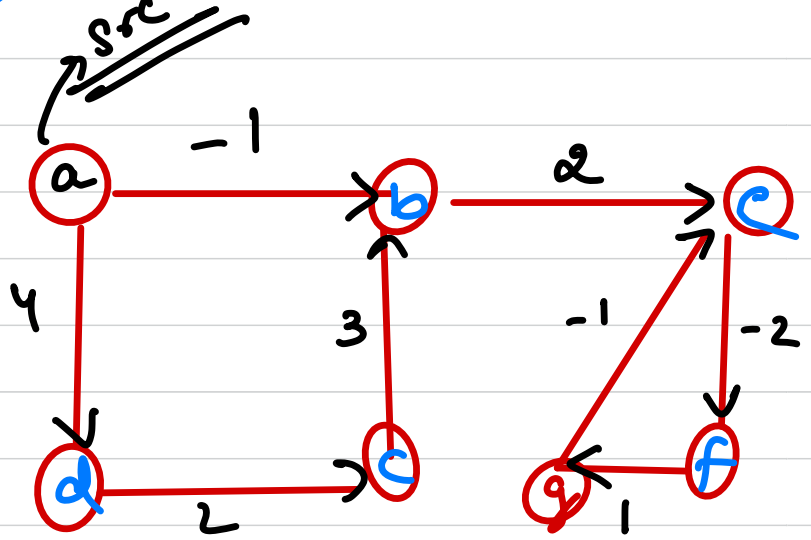
↓

3 and

Bellman ford → DP → SSSP

BU

V-1



relaxation

if(SP[u] > SP[v] + E(u,v))

SP[u] = SP[v] + E(u,v)

a	b	c	d	e	f	g
0	∞	∞	∞	∞	∞	∞
	-1	6	4			


```
for i = 1 — |V - 1|
    for each edge (u, v)
        relax (u, v)
```

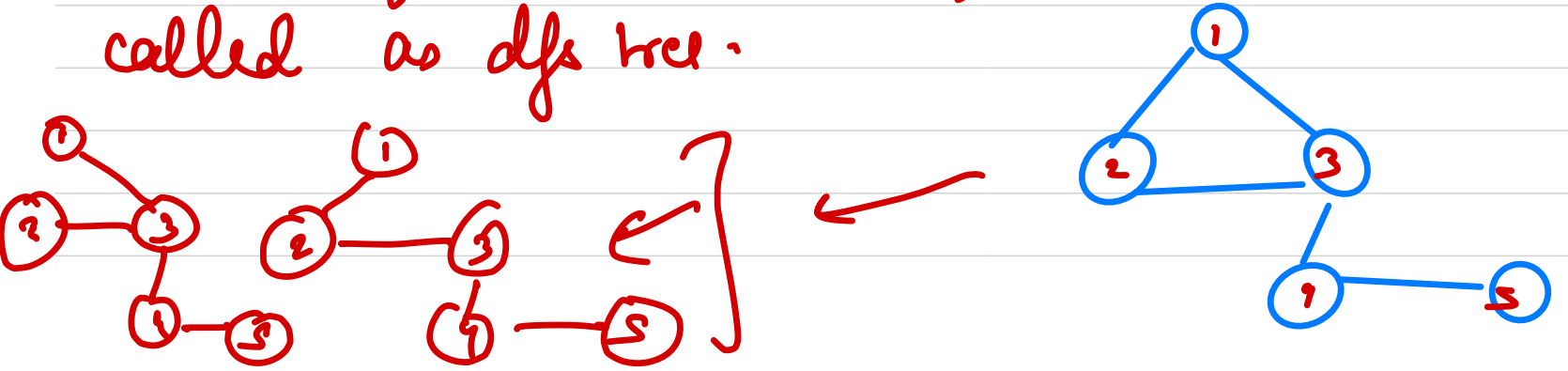
$f(x, y)$ = length of shortest path whose length is atmost y and path is from src to x

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

$u \rightarrow$ Edge wt

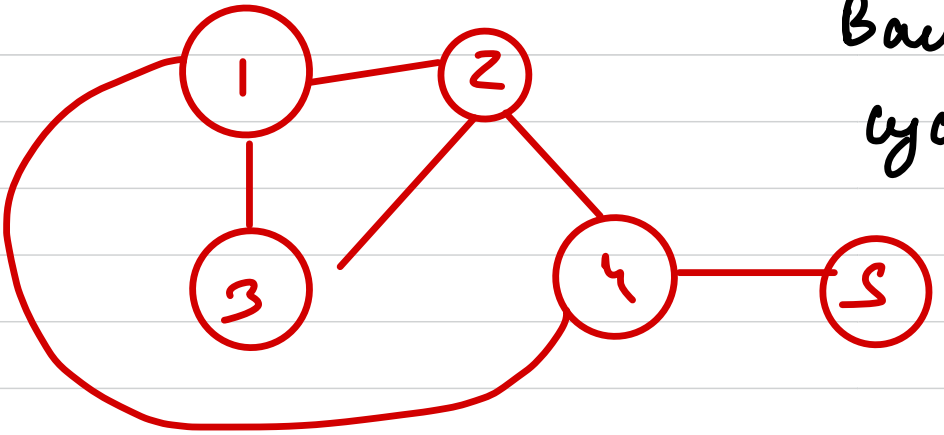
Articulation Points & Bridges

* 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.

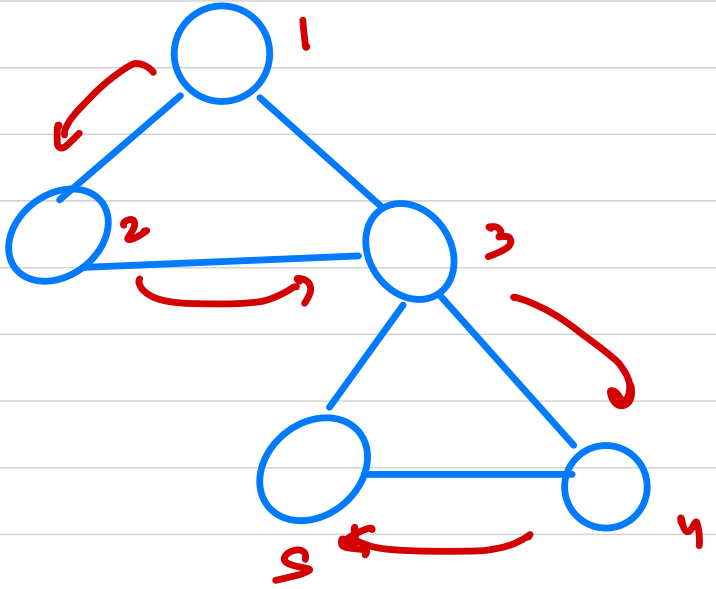
* Back Edge → So this edge connects a node x to an already visited node in graph.



Back edge leads to cycle in dfs tree.

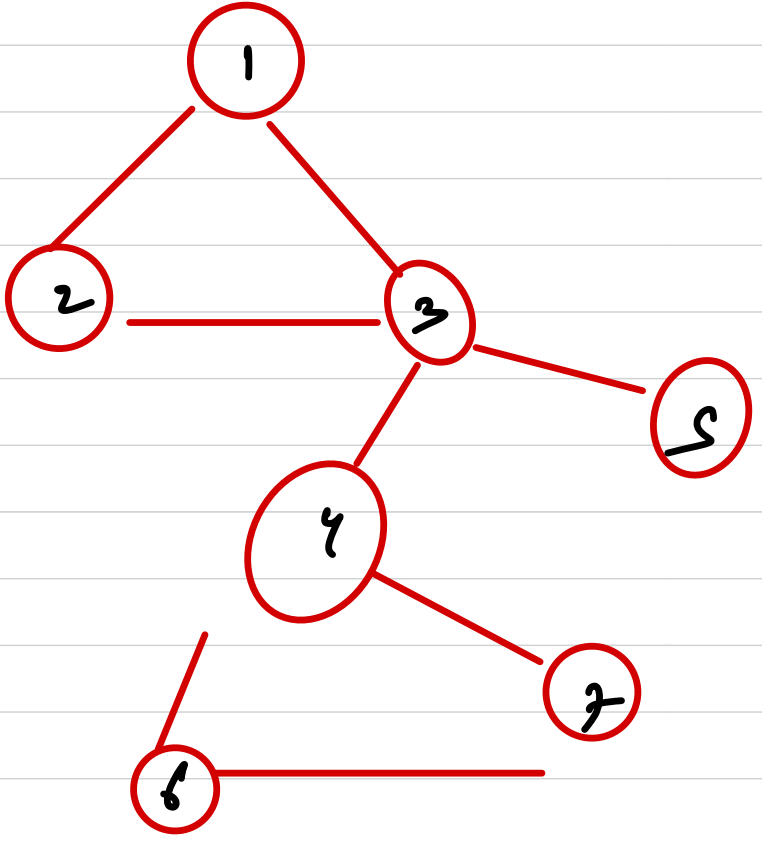
Span edge → edges included in dfs tree.

* Discovery time instance → the first moment in time when you visit a node in dfs.

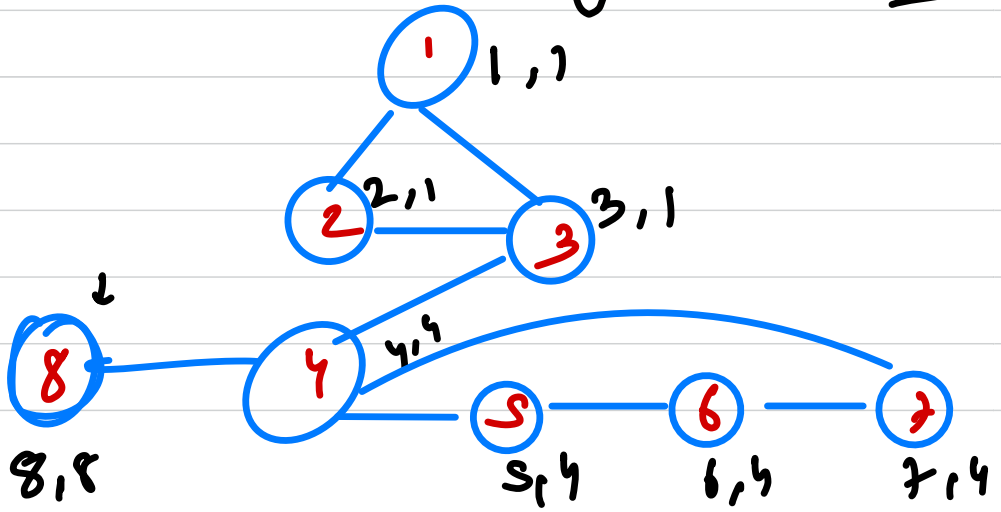


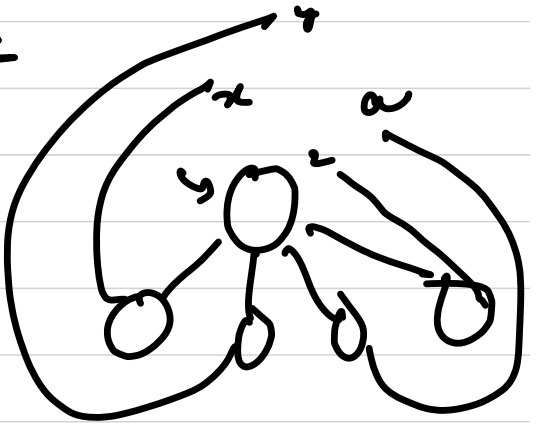
Articulation point → A vertex/ node in a graph is called articulation point, if by removing the node & edges corresponding to it, we split the graph into more components i.e. the no of connected components increases.

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

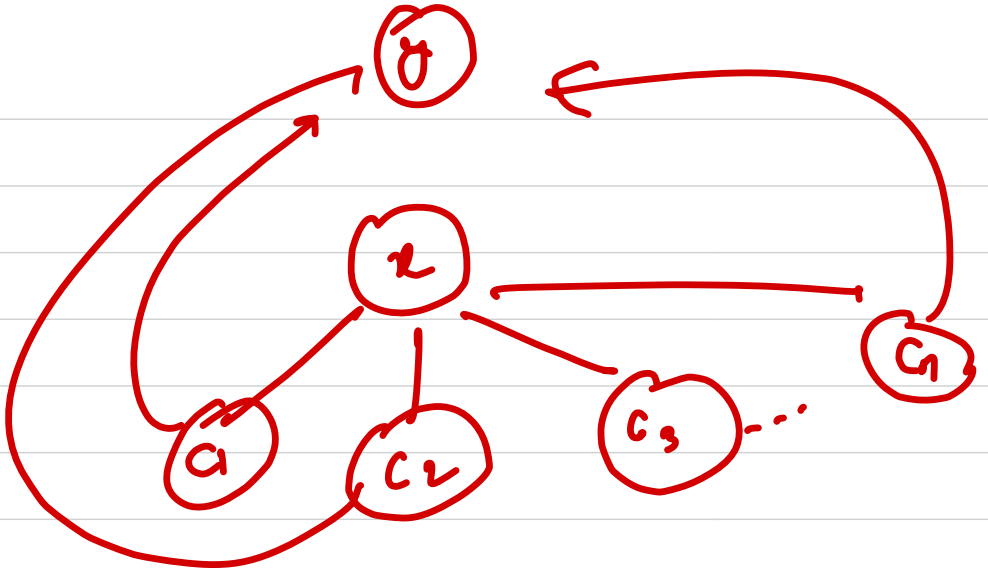

1
2
3
5
4
7
6

Lowest time unit → minimum discovery time of a node that any back edge is pointing in the subtree of current node.



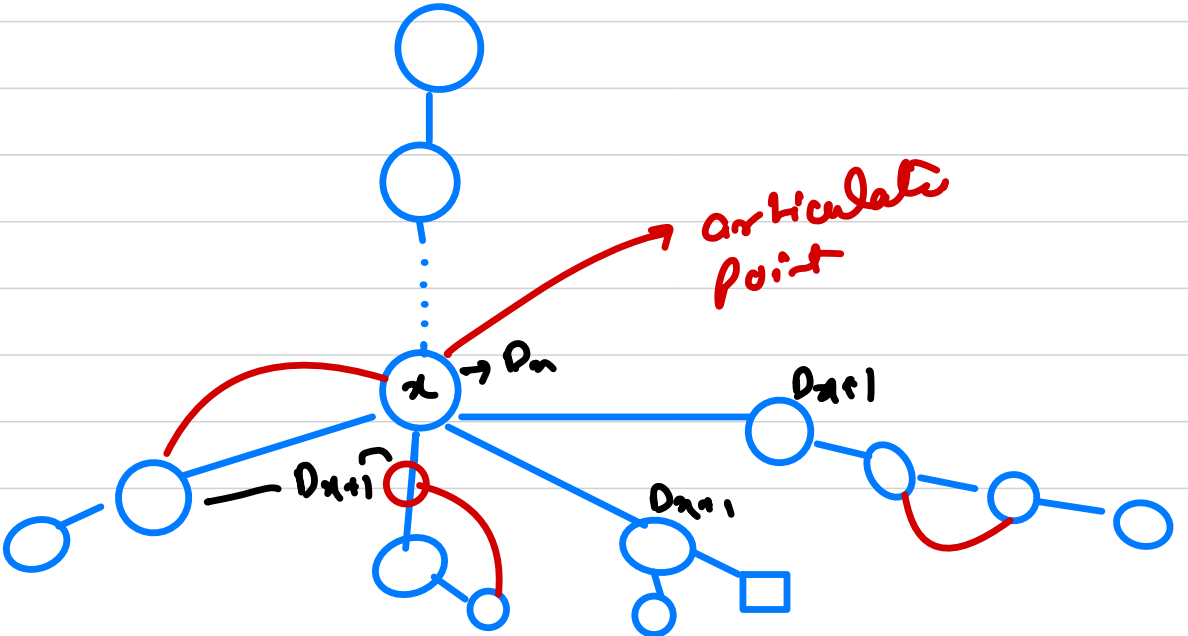


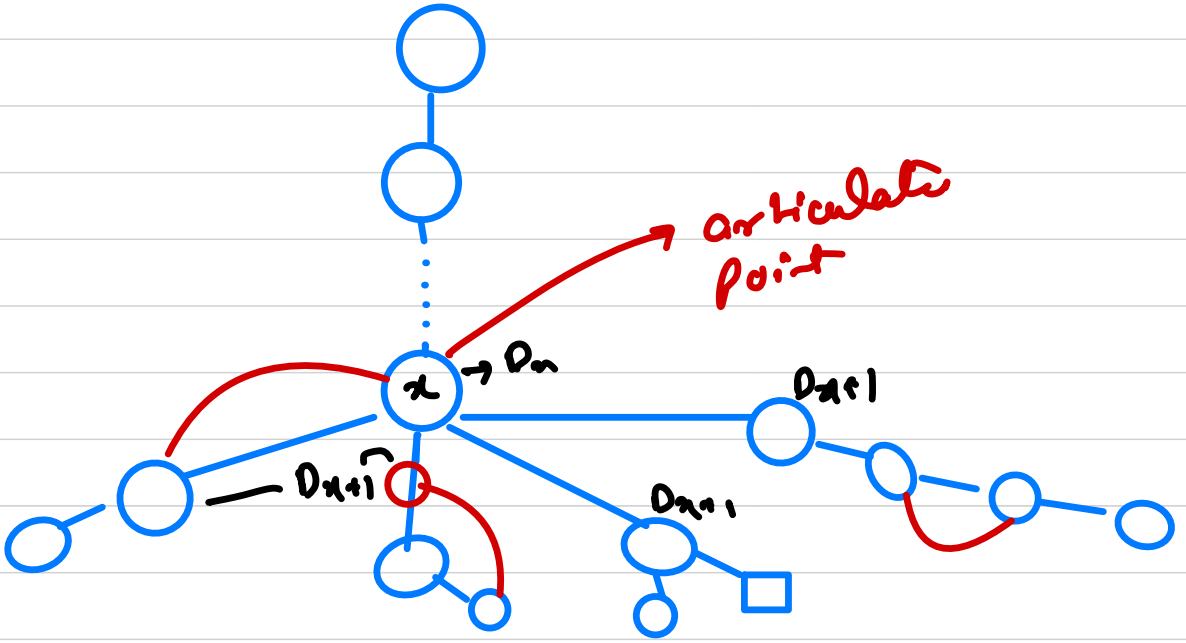
min(x, y, z, a)



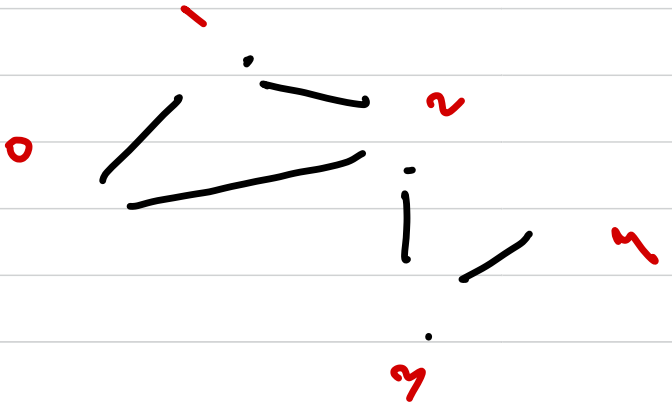
if any child of node x, if it has a back
/or lowest time
to a node whose discovery time greater or equal to x
then x is an articulation point

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



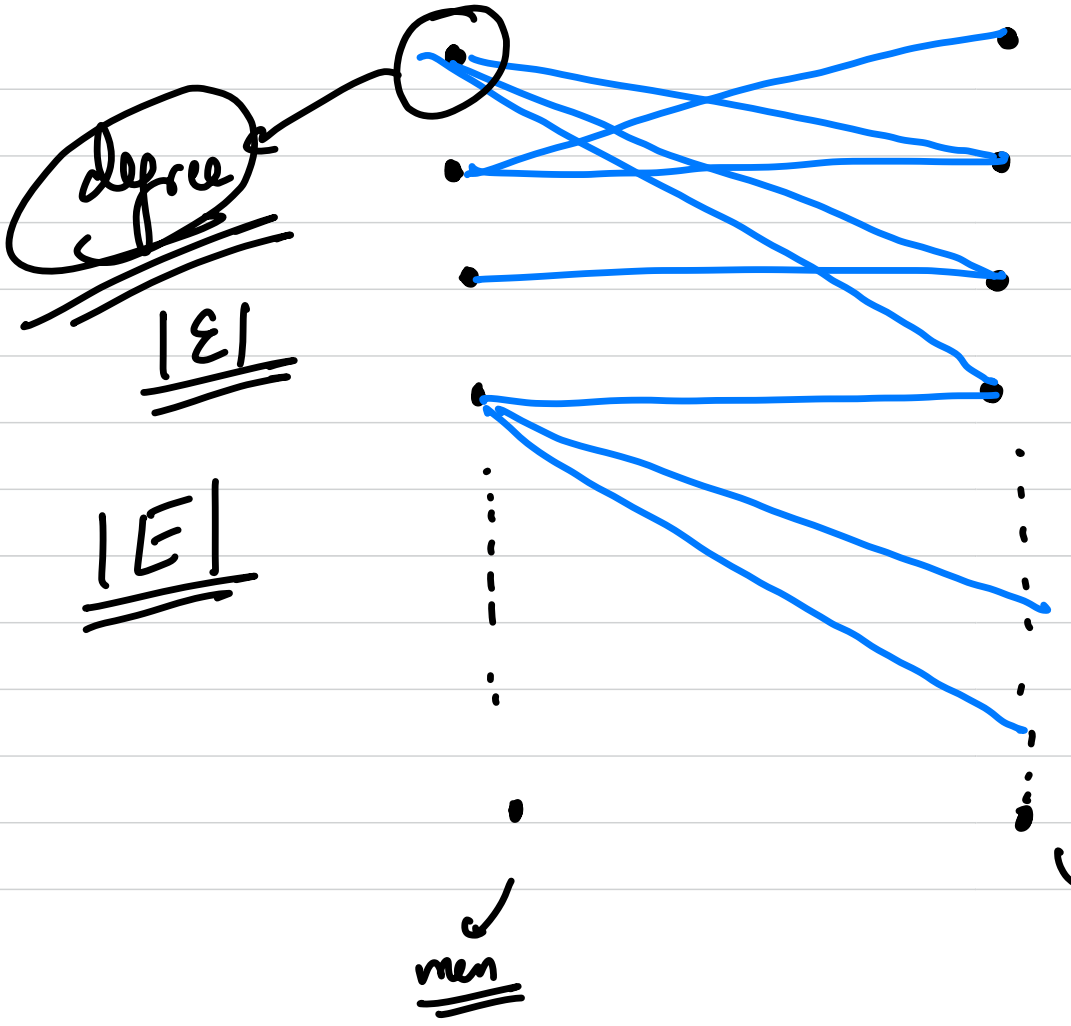


↳ strictly less


1
0
2
4
3

i.e. male & female count

Q ⇒ Let's say we have a gender distn of a county 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 partner for women. find x/y.



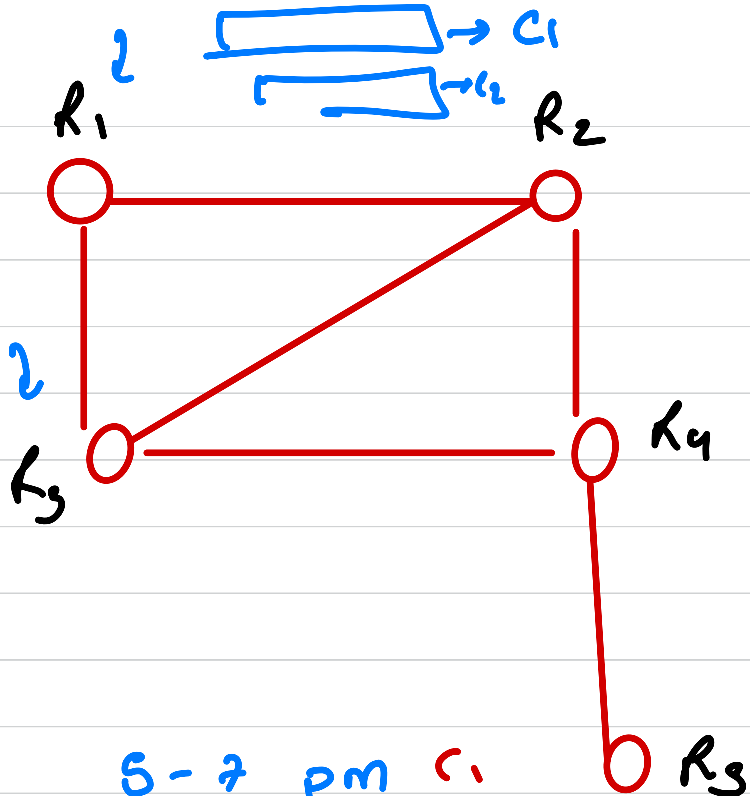
$$x = \frac{\sum_{x \in men} deg(x)}{|men|}$$

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

$$\frac{x}{y} = \frac{\cancel{|E|}}{\cancel{|men|}} \Big/ \frac{\cancel{|E|}}{|women|}$$

$$\frac{x}{y} = \frac{|women|}{|men|}$$

~~meetings~~



graph colouring problem

$X(G)$

Range of meeting times

5 - 7 pm c1
7 - 9 pm c2
9 - 11 pm c3
11 - 2 am c4
3 - 5 am c5

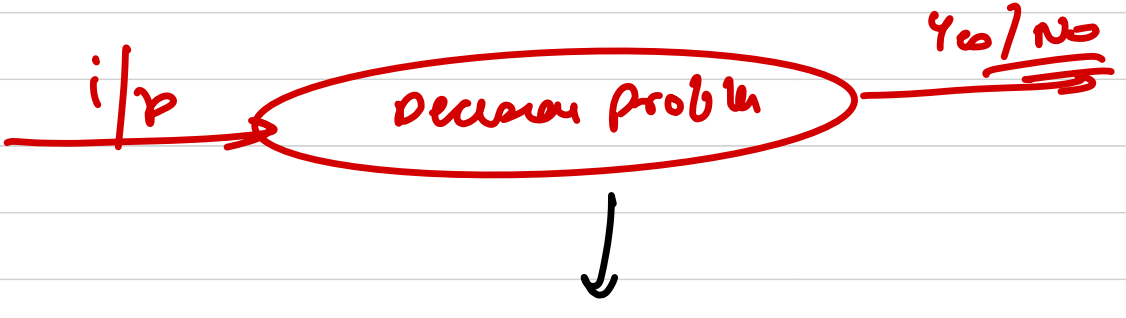
x → no. of meetings

What is a graph colouring problem??

Given a graph G, 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. $\chi(G)$ → min value of k.

→ Decision problem → yes/no ans associated



Decision procedure

problems → algorithm

solvable

desired result
expected time

(1) P- class → set of decision problem which can polynomial time solvable. $O(n^k)$

k → const

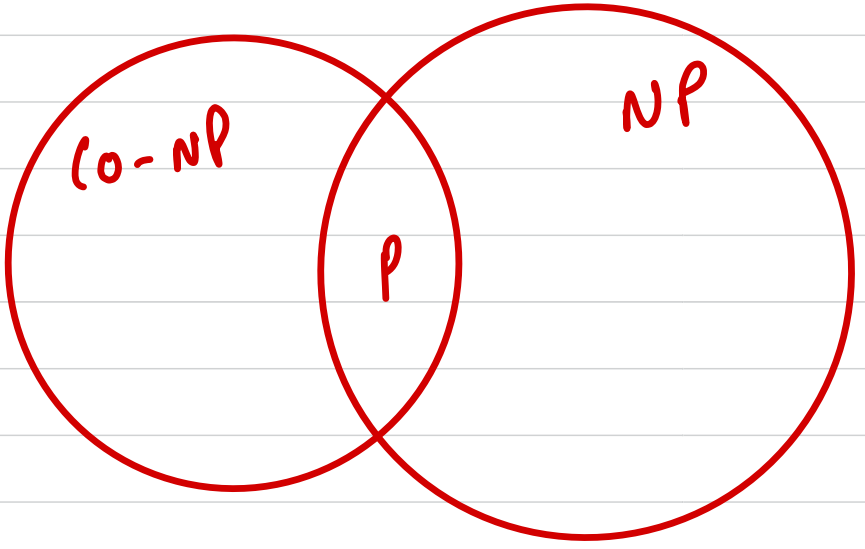
NP class → non-polynomial time class →
set of decision problem that are only
verifiable in polynomial time:

Ex 500 students → 100 rooms
|→ few students should not
be grouped together.

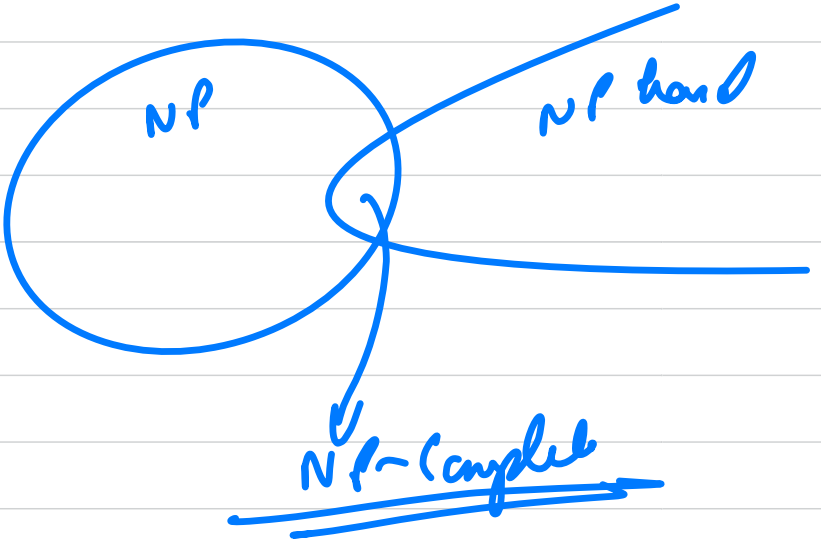
Co-NP → "No"

np-hard

deter
non-det

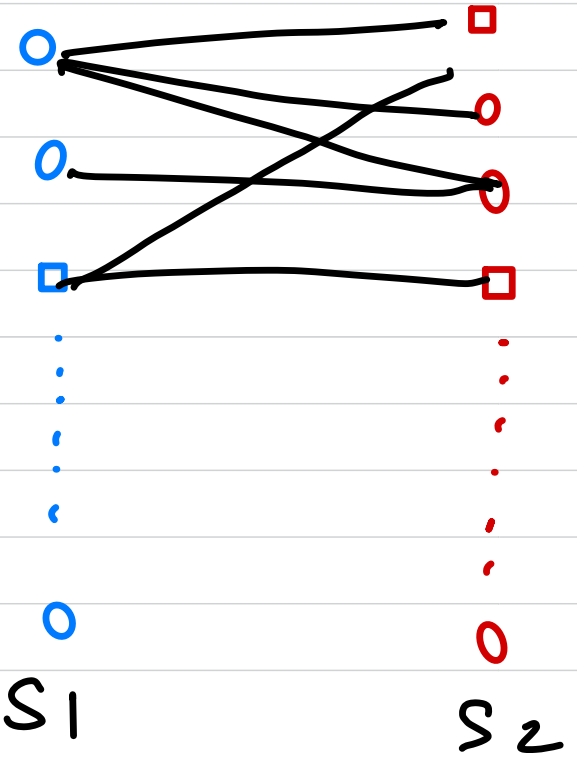


NP complete → Then lie in both NP & NP-hard

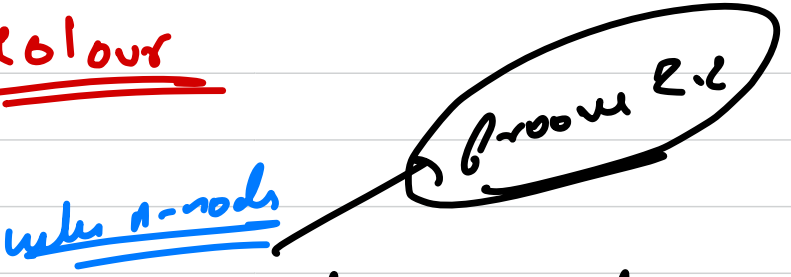


★ Bipartite graph → 2 colourable graph

DSU <.2



→ algorithm → colour



Theorem Given a graph G has a degree d, then the basic algo uses at most $d+1$ colors for G.

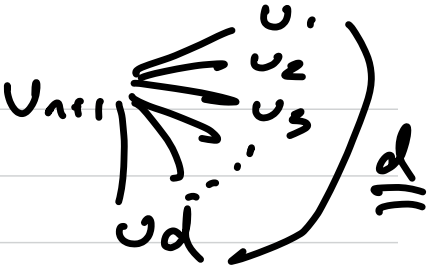
Using Induction

I.H → $P(n)$ ← true for n nodes

BC → $n=1$

IS → P(N) clearly true

Proof → P(n+1) → true



$G_n \to (V, E)$ → (n+1) node graph. & D is the max degree in G_n.

v_1 v_2 - - - - - v_{n-1} v_n v_{n+1}

Remove v_{n+1} & corresponding edges, & make a new graph $G' \leq d$

To colour G' add to P(N) → we need d+1 colors at most.

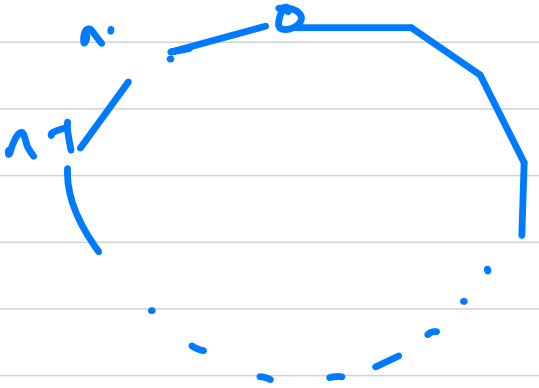
$G' \to G \to (d+1)$ colors H!

Given a graph detect if it has odd length cycle?:

2 HW

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

(theorem)



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

Part 1 → G is bipartite $\Rightarrow$ G has no odd cycle

ans → Suppose G is bipartite

$V(G) = X \cup Y$ and $X \cap Y = \phi$

for every edge $e \in E(G)$ $e = xy$ where $x \in X$, $y \in Y$

Proof by contradiction:

Assume G has atleast one odd cycle.
of odd length

Let C be a cycle, such that

$C = \{ v_1\ v_2\ v_3 \ldots\ldots\ v_n \}$ where n is odd

$v_1 \in X$ 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 $v_1 \in X$ & $v_n \in X$
$v_1 v_n \in E(G)$ $\longrightarrow$ This is impossible.

Part 2: → If a graph has no odd cycle ⇒ it is bipartite

⇒ Suppose all cycles in G is of even length,

→ if graph is disconnected, then we consider our proof for any component

Let say $m \in V(G)$

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

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

(X, Y are 2 partite set)

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

$X \cap Y = \phi$

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:

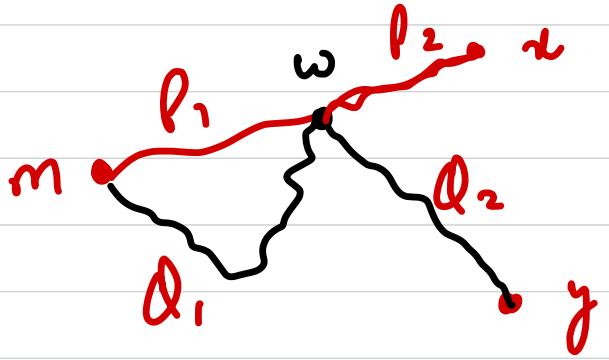
assume → that $x, y \in X$ or

$x, y \in Y$

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 $m \to w$
p_2 : Shortest path from $w \to x$

$p = p_1 p_2$

q_1 : Shortest path from $m \to w$
q_2 : Shortest path from $w \to y$

$q = q_1 q_2$

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

P_1 and Q_1 are the shortest paths weak w from m.

$$|P_1| = |Q_1|$$

Now, $|P_2|$ and $|Q_2|$ will have same parity.

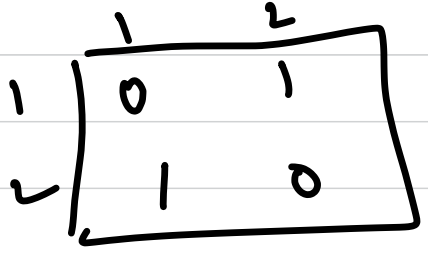
So both P_2 & Q_2 are odd or both are even

Then the path from $x \xrightarrow{P_2^{-1}} w \xrightarrow{Q_2} y$ is always even parity.

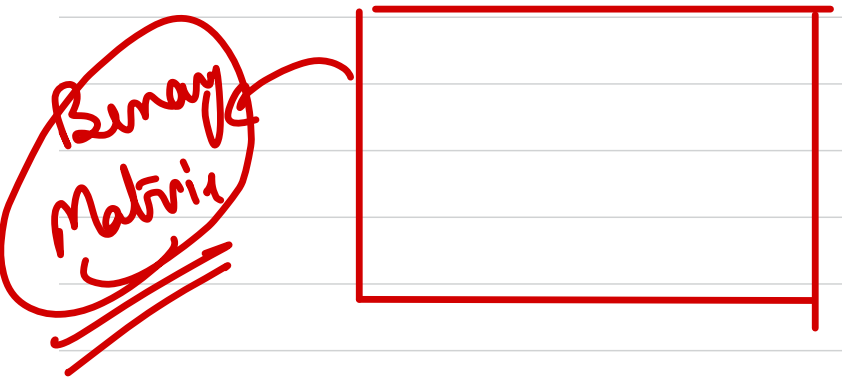
Now if $x, y \in \bar{E}(G)$ then $P_2 \sim Q_2$ together with xy will form a cycle of odd length.
This is contradicting as $x, y \notin \bar{E}(G)$

Q: Given a matrix M of binary values. find if we can have a 1D array B such that

$$M[i][j] = |B_i - B_j|$$



Can we treat 0's & 1's differently



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

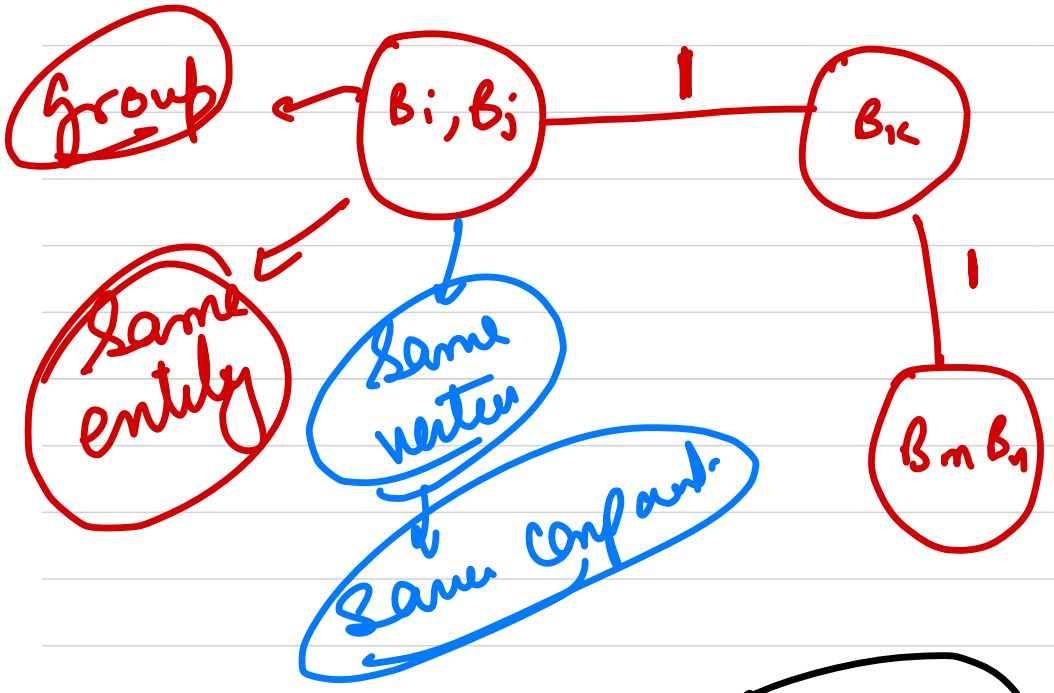
is there an array B, such that

$M_{ij} = M_{ji}$ (observe)

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

$M_{ii} == 0$ → false → false

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

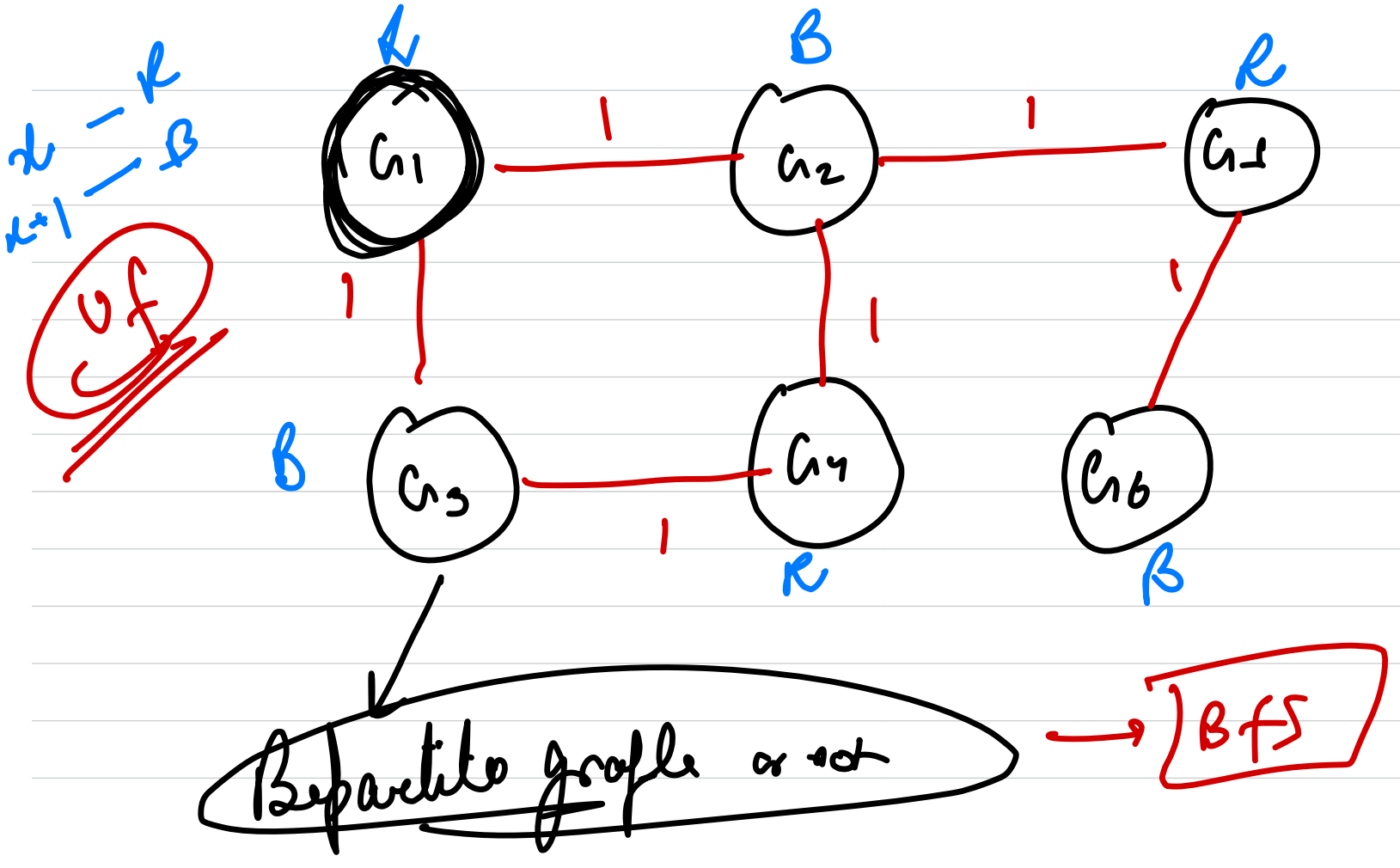


$M_{ik} == 1$

$B_i != B_k$

$M_{ij} == 0$

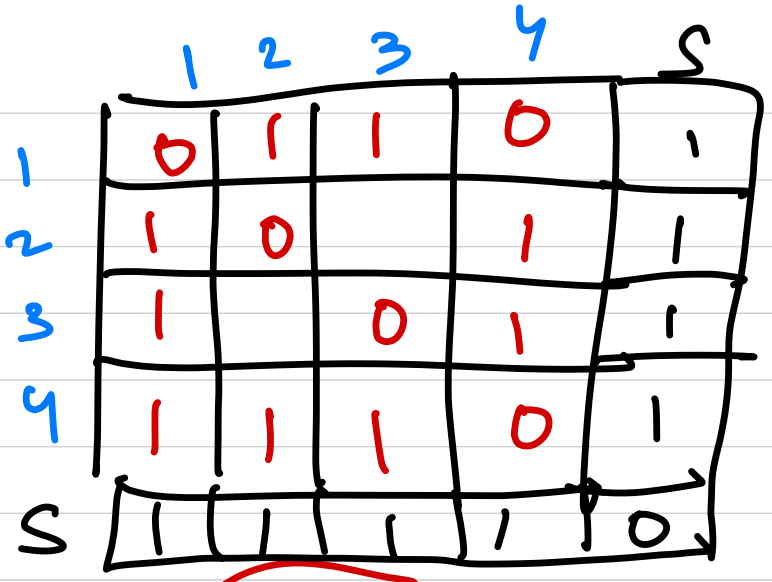
$M_{xy} == 1$

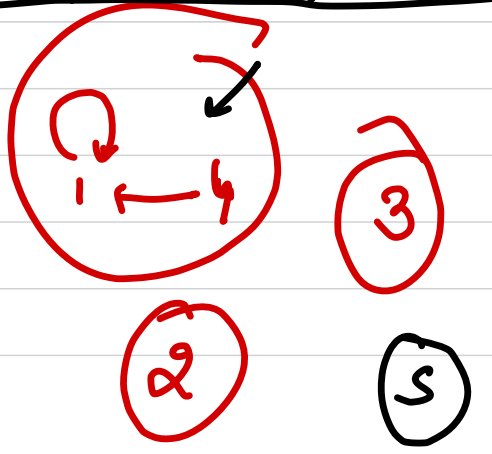

R
B
R
1
1
1
1
1
B
R
B
Bipartite graph or not
BFS

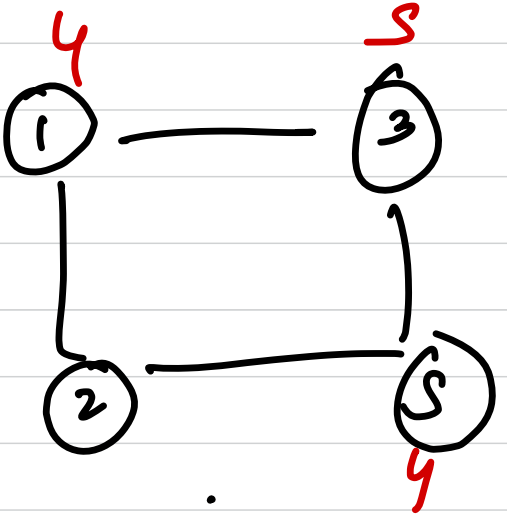
→ Using DSU, merge all the nodes in the corresponding groups

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

→ Check if it is Bipartite


1 2 3 4 S
1 0 1 1 0 1
2 1 0 1 1
3 1 0 1 1
4 1 1 1 0 1
S 1 1 1 1 1 0


1 ← 4
3
2
S


4
S
1
3
2
S
4

⇒ 1 2 3

1 2

2 1

1 1 → ew

1 2 → odd ✓

2 3

1 3 → ew

3 2

2 1 → odd ✓

1, 2

2 2 → ew

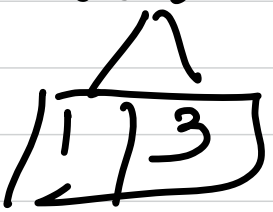
3, 2

2 3 → od ✓

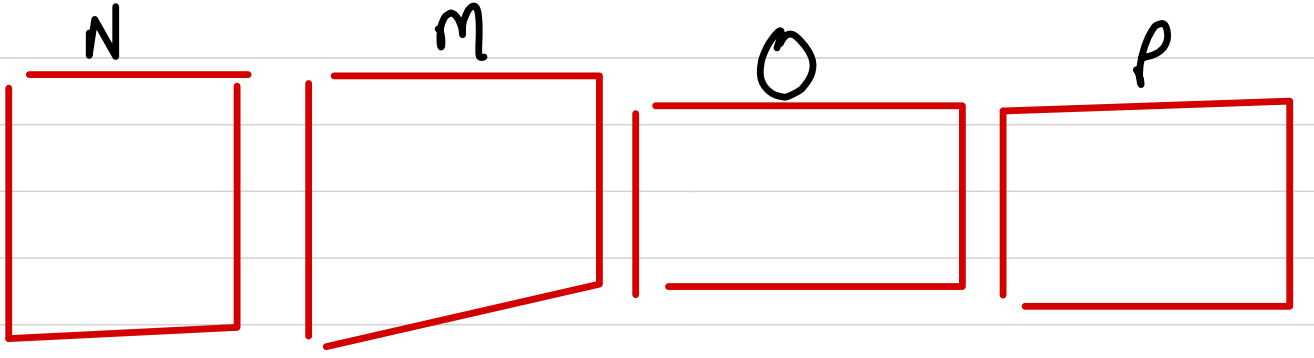
3 1 → ew

2 2 → od ✓ 3 3 → ew

⇒ odd + even → odd



⇒ disconnected graph



product of N, M, O, P

How to solve for one Component

Partite sets

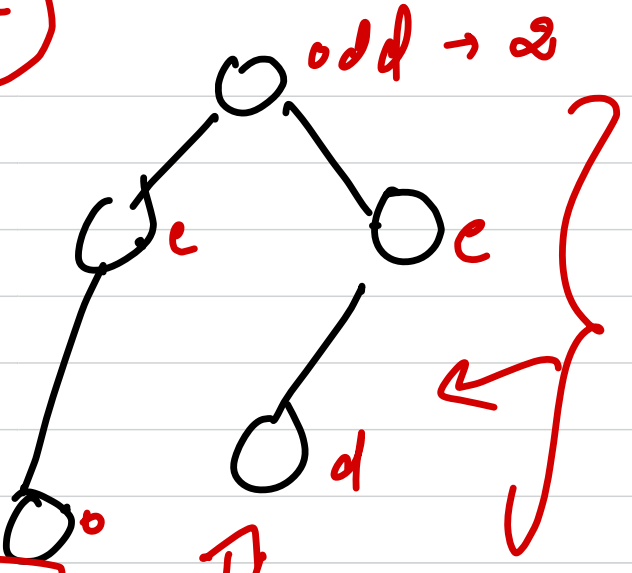
odd → 2

even → 2

odd → 1, 3

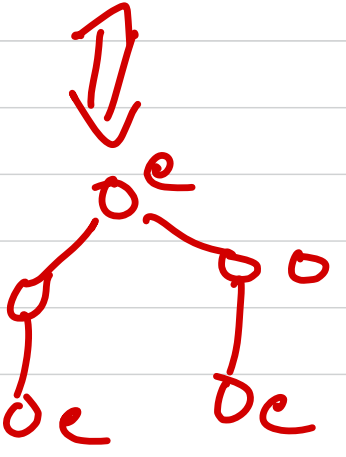
B

Bipartite graphs



$2 \times 1 \times 1 \times 2 \times 2$

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

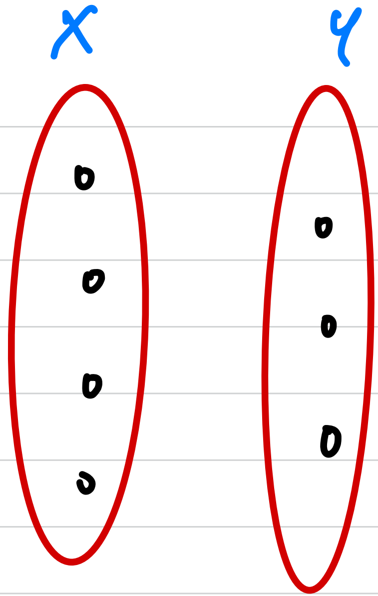


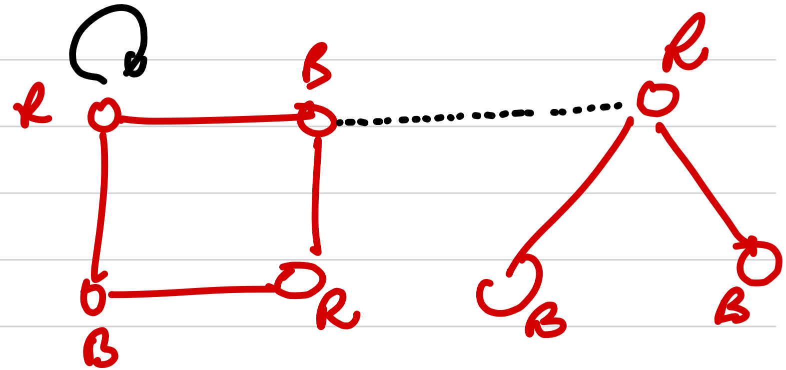
count_R

count_B

$$cmd_t \rightarrow 2^{cmd\text{-}k} + 2^{cmd\text{-}b}$$

Lemma





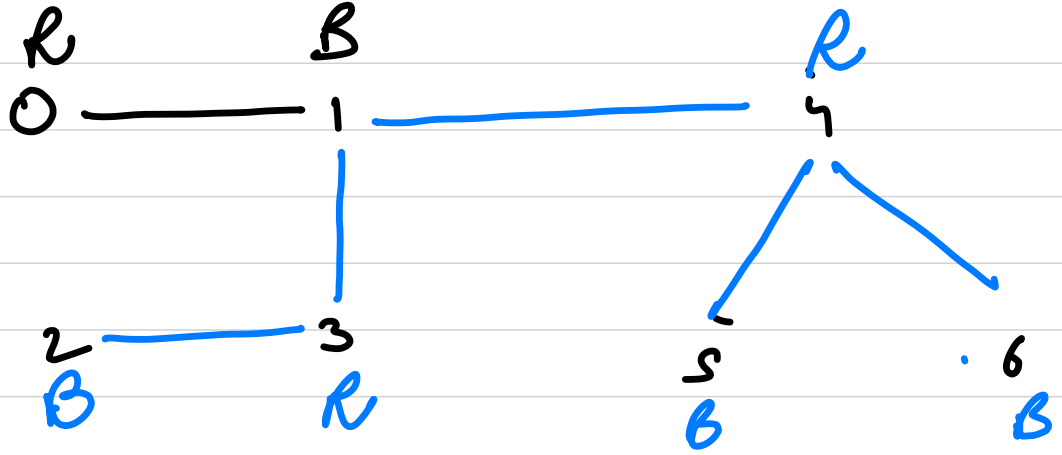
Rem [x] → lemma for x in P[x]

2)

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

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

graph



R → even

B → odd

check Bipartite → O(log *V)