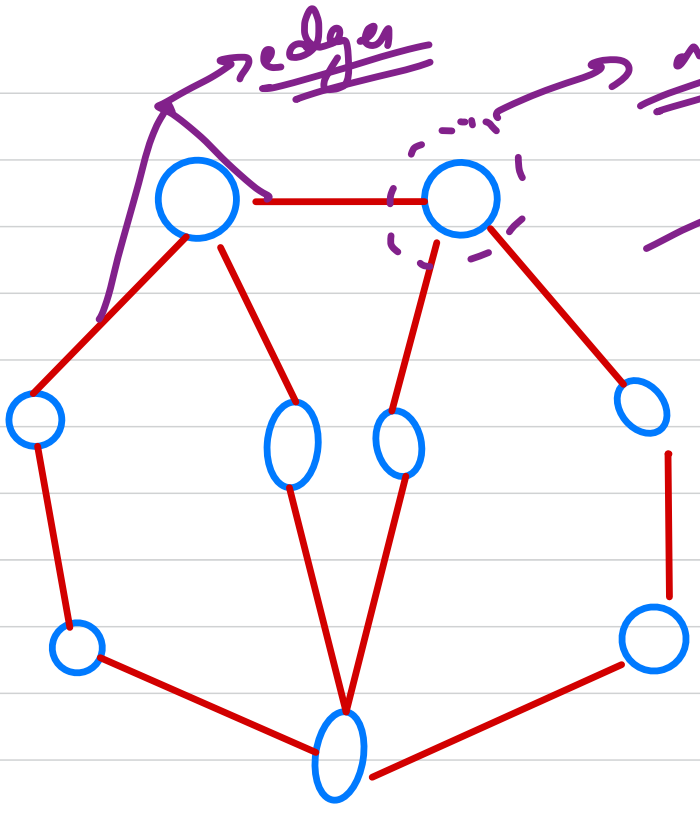
Graphs

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.

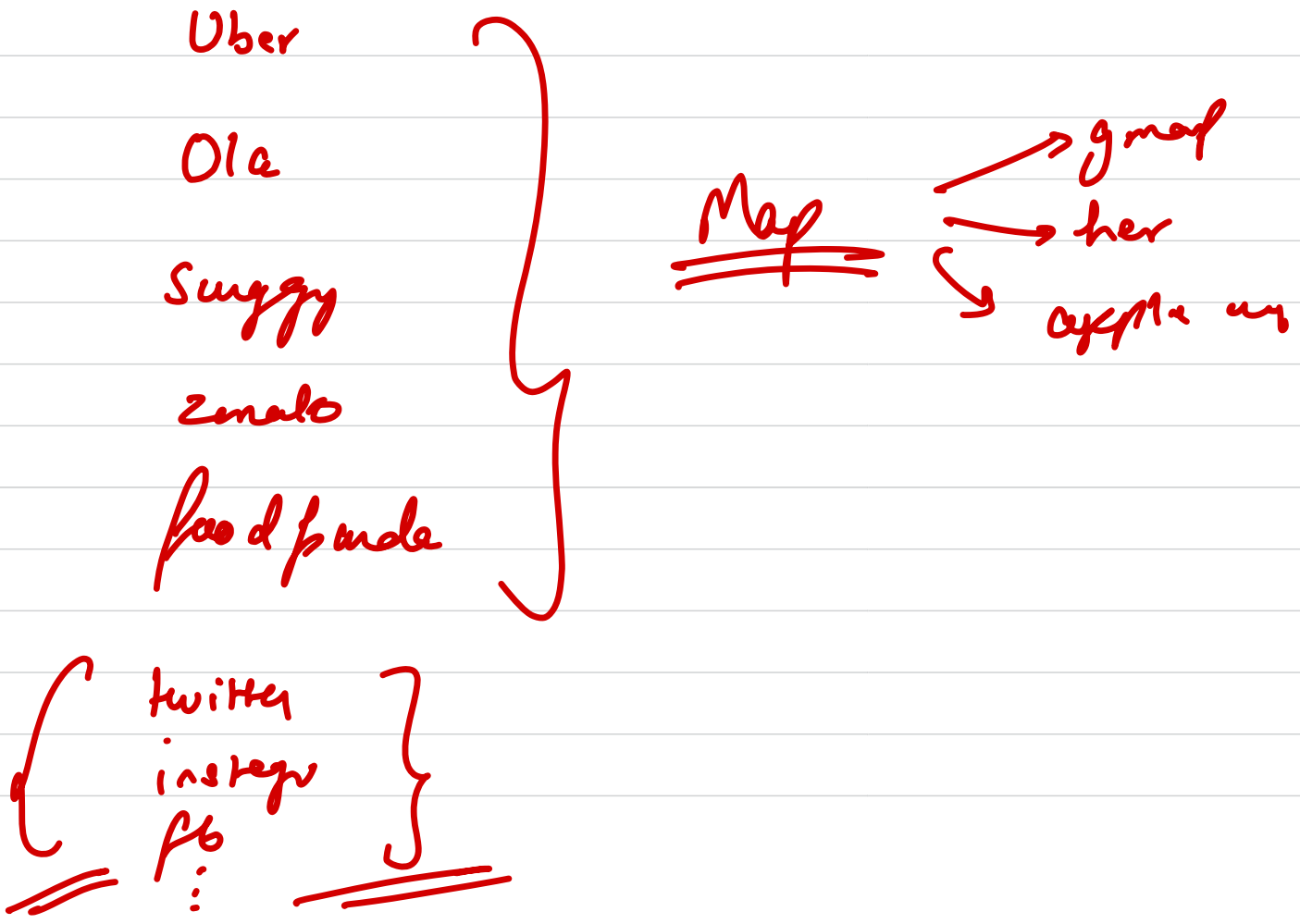


A typical graph

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

Electrical → Circuit Organization

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

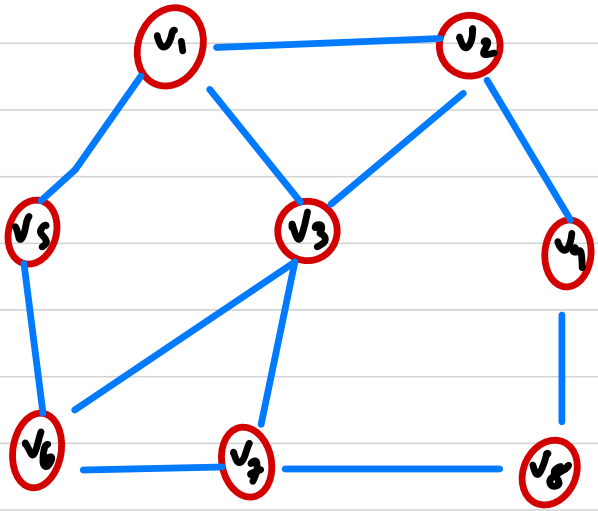

Uber
Ola
Swiggy
Zomato
food panda
Map
gmap
twitter
insta
fb

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

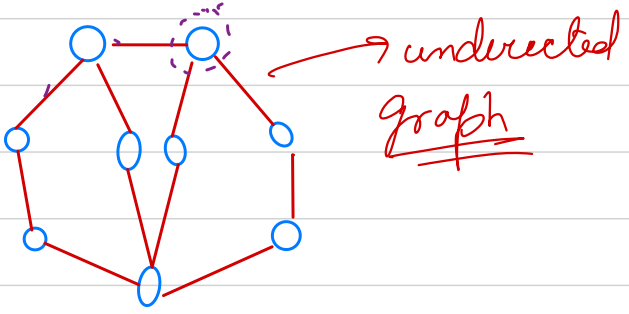
$\rightarrow G = (V, E) \rightarrow$

mathematical notation of a graph

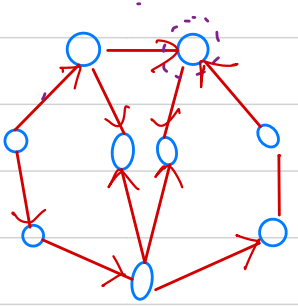
A graph G is an ordered pair of a set V vertices & E, a set of edges.

Types of graph :

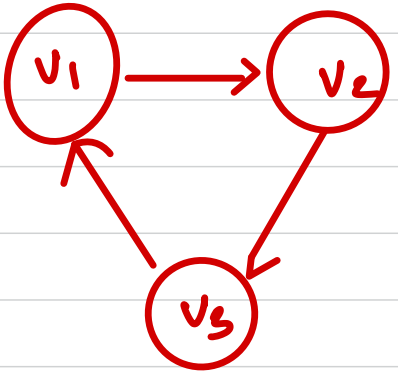
1. Undirected → Facebook
2. Directed → Instagram, Twitter



→ undirected graph



→ Directed graph



$G = (V, E)$

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

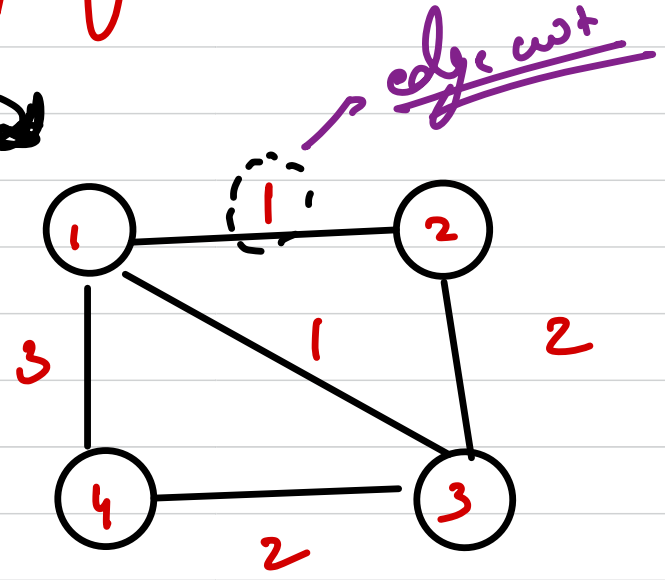
$\{v_2, v_3\}$

$\{v_3, v_1\} \}$

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

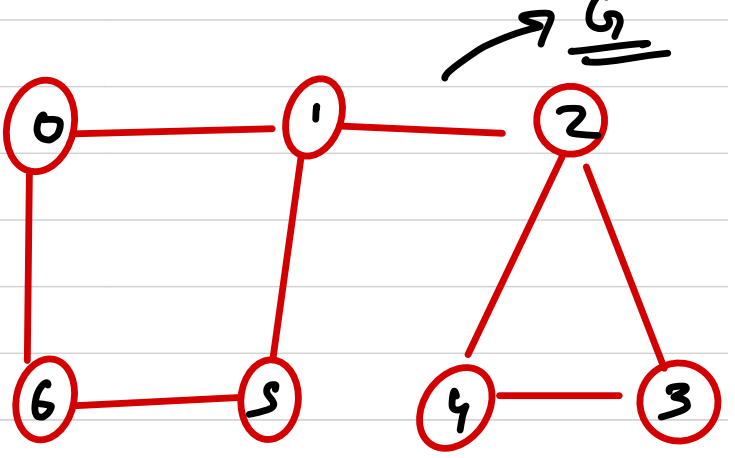
(2) Adjacency List ⟶ Adjacency Map

(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

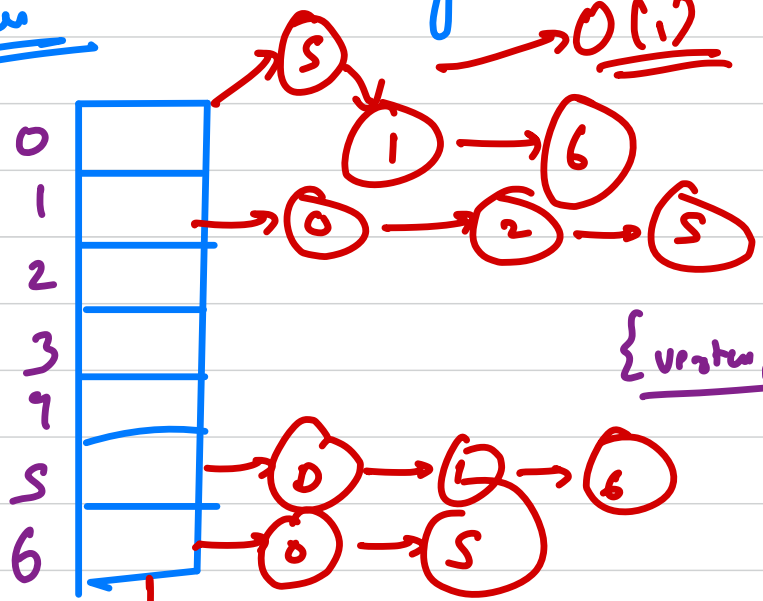
$|V| \Rightarrow$ no of vertices

Adjacency list

Array of LL → space option

↳ edges

vertices



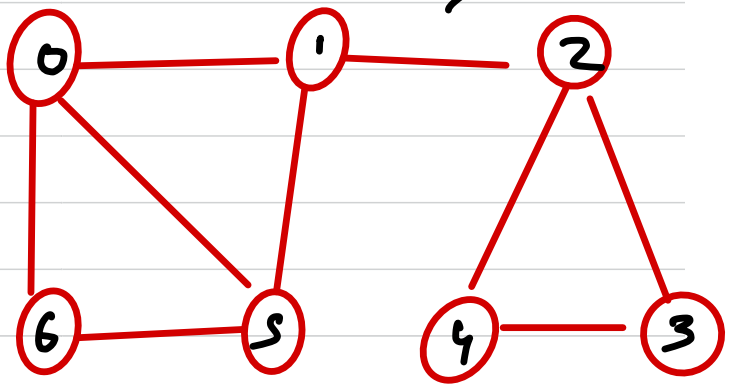
→ O(1)

Bucket array → i^{th} index stores info about i^{th} edge

add at head

→ good for sparse gr

→ G



{vertex, wt} → node

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

adjacency map

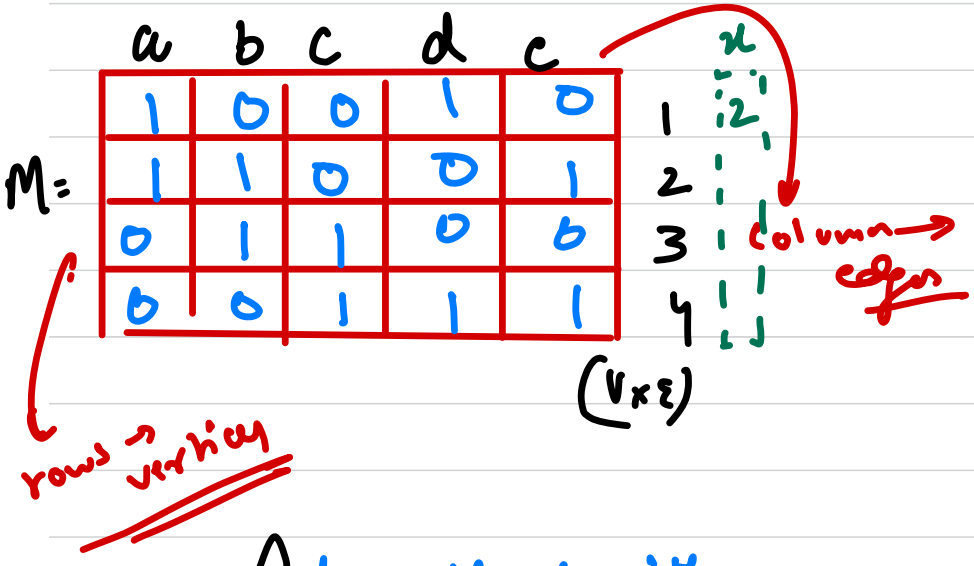
array of hashmap

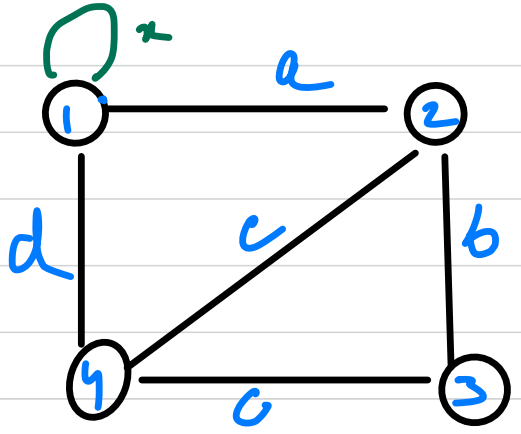
hasedge (u,v) → O(1)

Incidence Matrix

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

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





$(V \times E)$ dimension

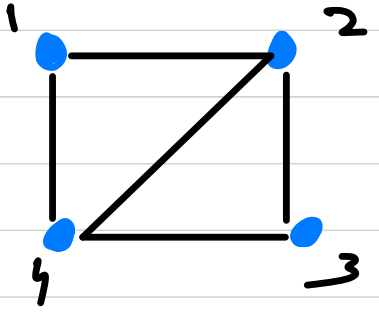
$$M_{ij} = \begin{cases} 1 & \text{if the } i^{th} \text{ vertex belongs to the } j^{th} \text{ edge.} \\ 0 & \text{else} \end{cases}$$

rows_sum = degree

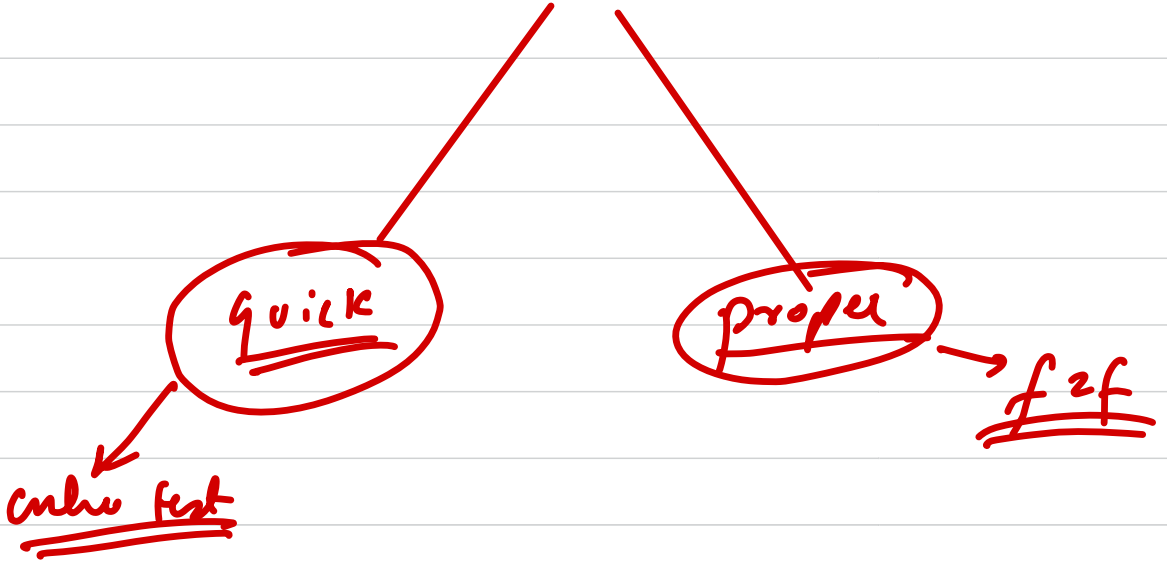
column_sum = 2

Q → What is a degree ??

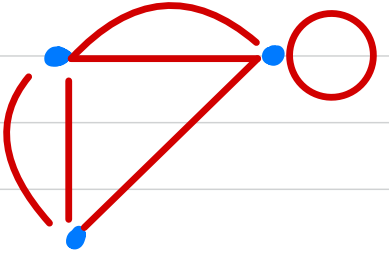
Degree of a vertex in a graph G, is the total no. of edges incident to it / associated with it (undirected graphs)



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

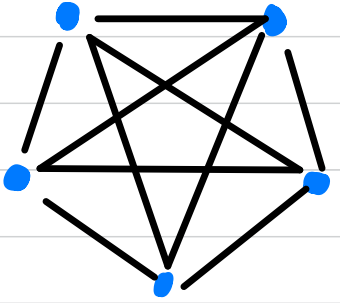

Quick
online test
proper
f2f

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



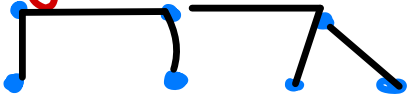
Simple Graph → An undirected graph in which both multiple edge & loops are not allowed.

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

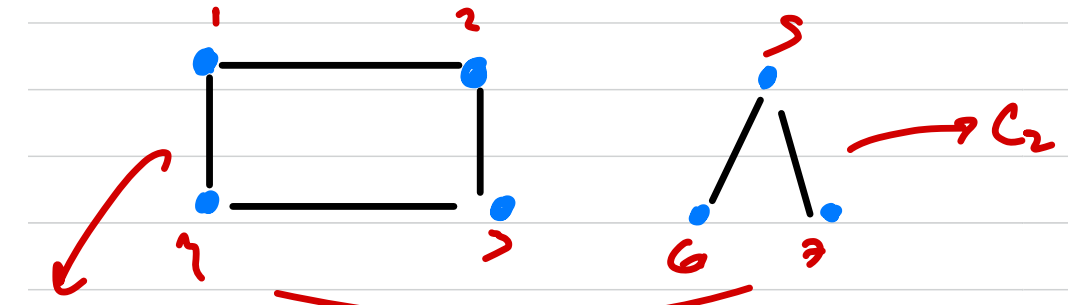


↙ complete graph

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



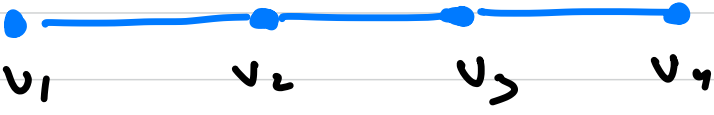
Disconnected graph → at least 2 vertices not have a path to every other vertex.



$G_1 = (\{1,2,3,4,5,6,7\}, \{\{1,2\}\{2,3\}\{3,4\}\{4,1\}\{5,6\}\{6,7\}\})$

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



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

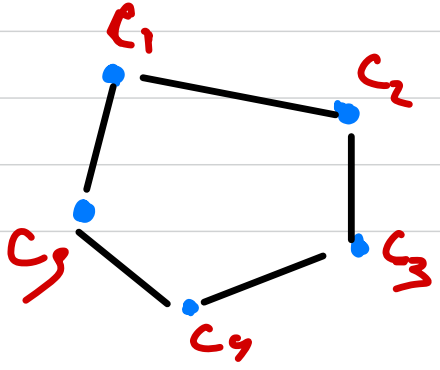
such that edge set of a graph is

$$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

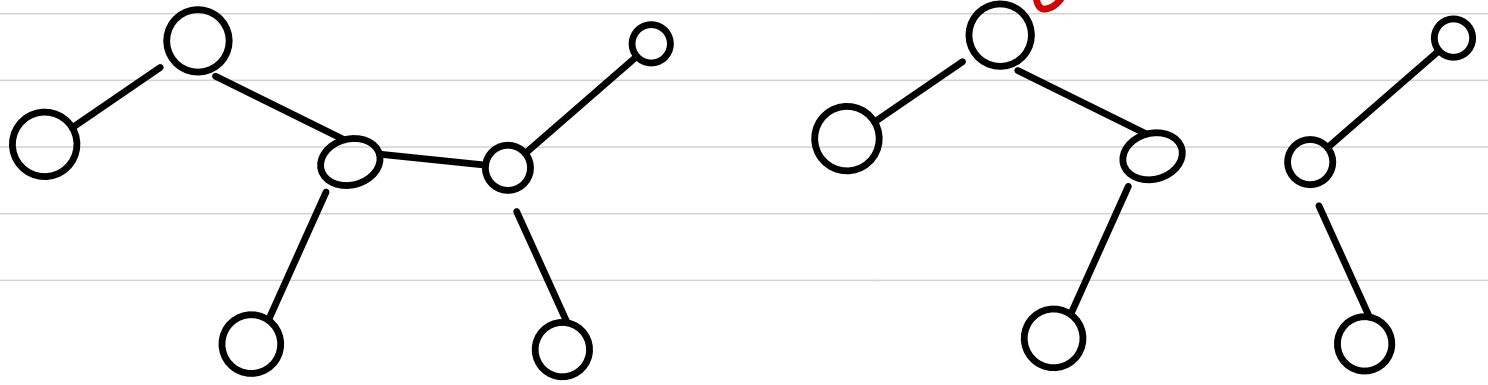
$V = \{v_1\ v_2\ v_3\ v_4\ \ldots\ v_n\}$ Such that edge set is

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

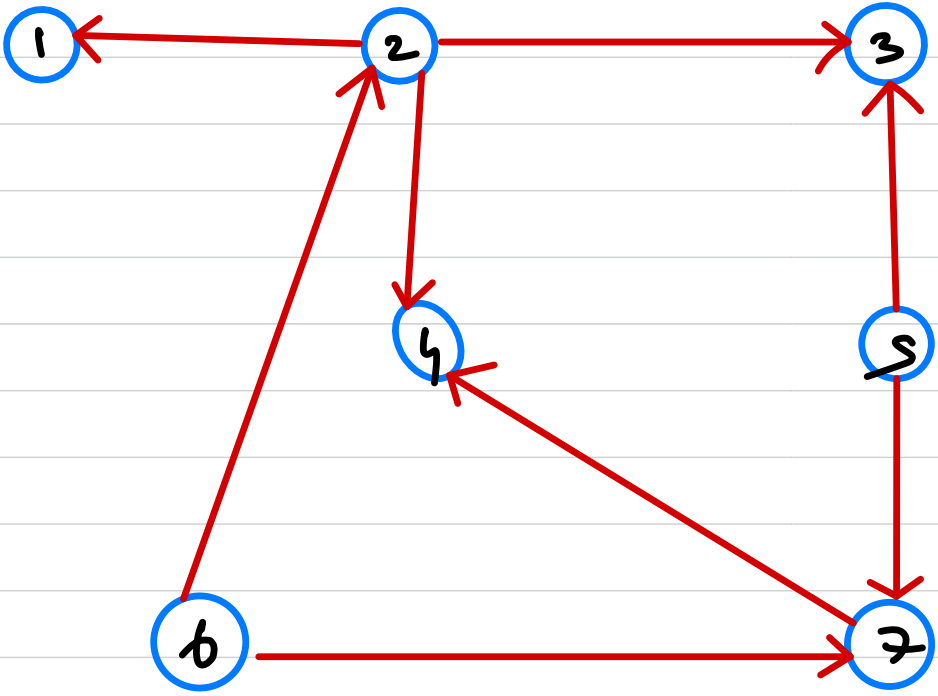


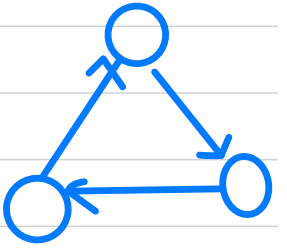
TREE → Tree is a connected graph with no cycles.

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

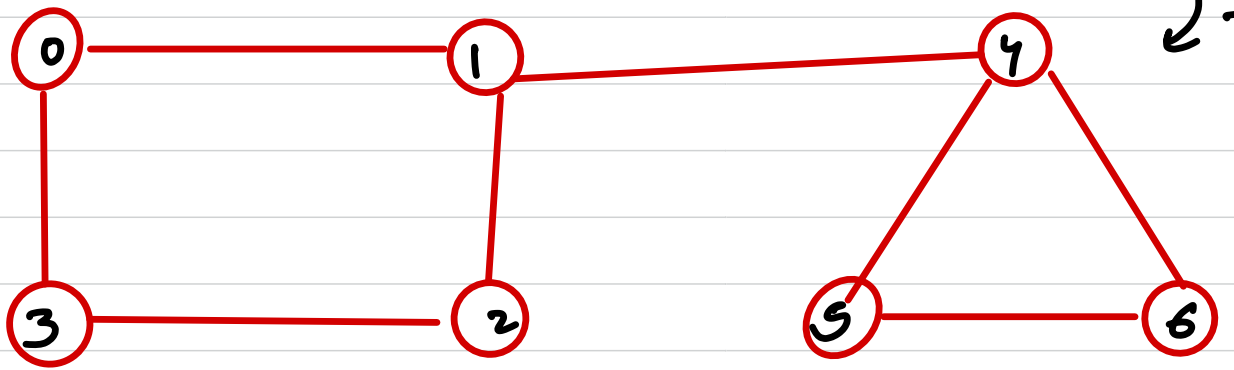
Depth first Search

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

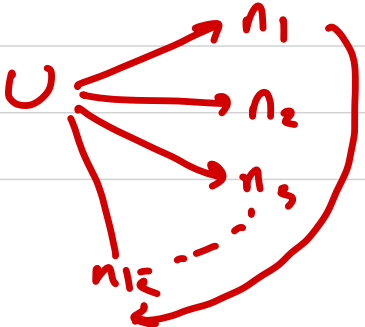
$f(u, v)$

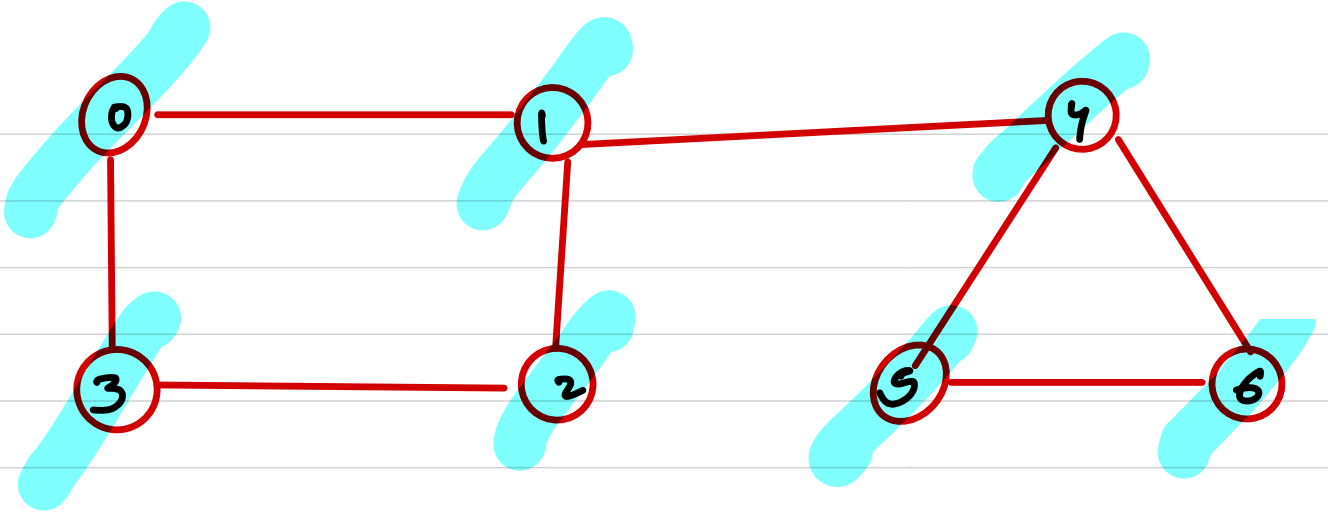
$=$

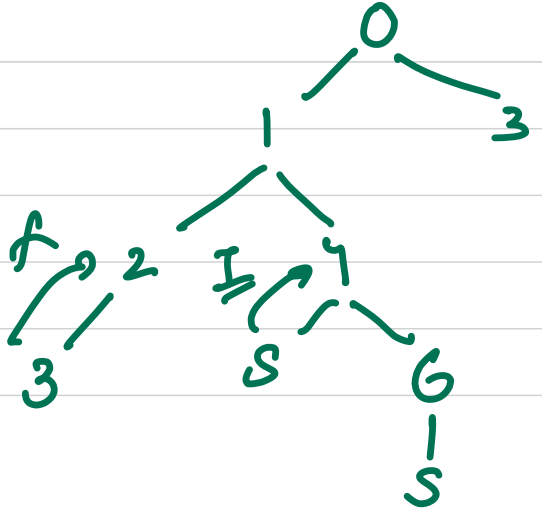
$\begin{cases} f(n_1, v) \\ \text{or} \\ f(n_2, v) \\ \text{or} \\ f(n_3, v) \\ \vdots \\ f(n_k, v) \end{cases}$

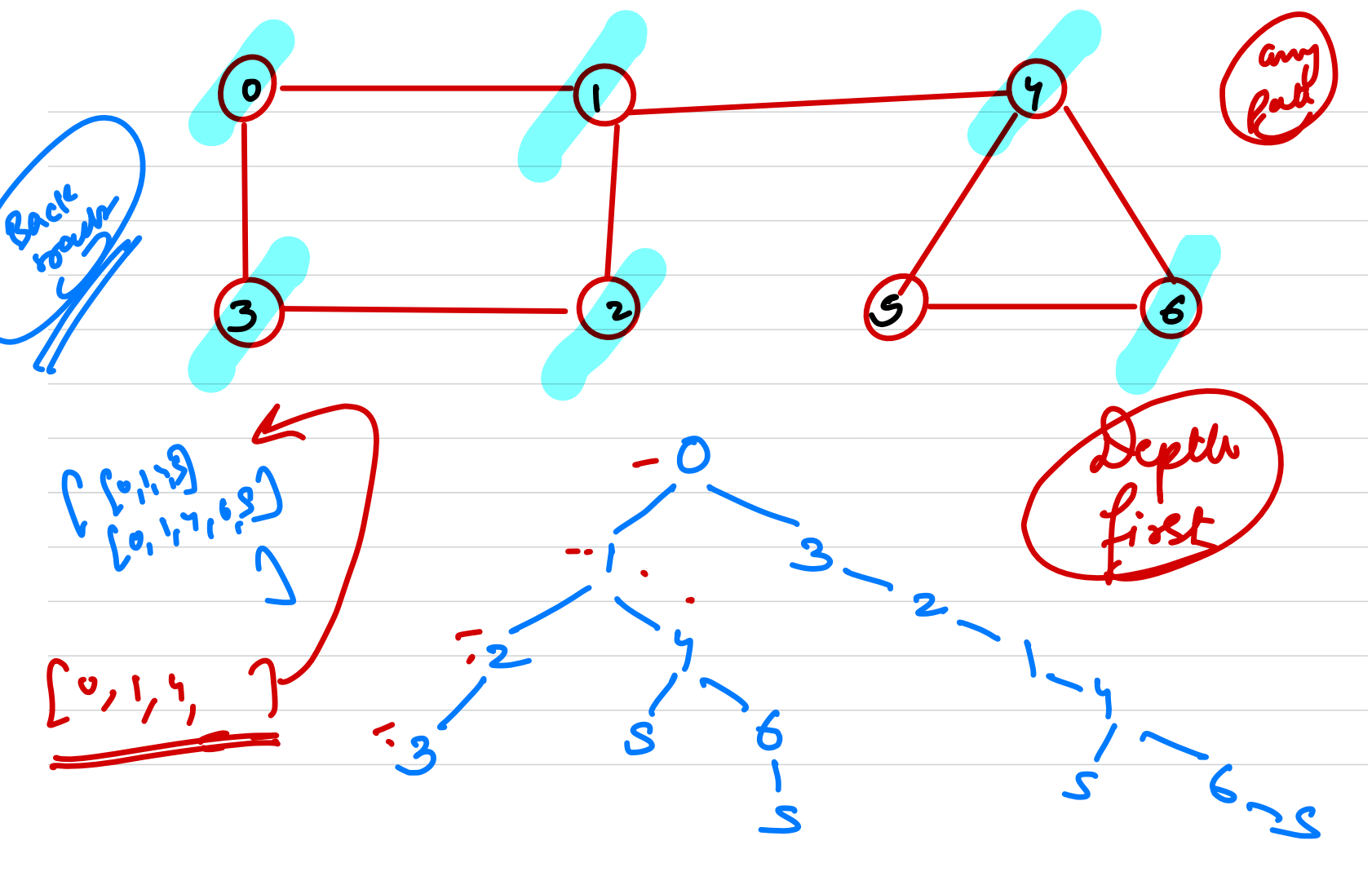
$\in$

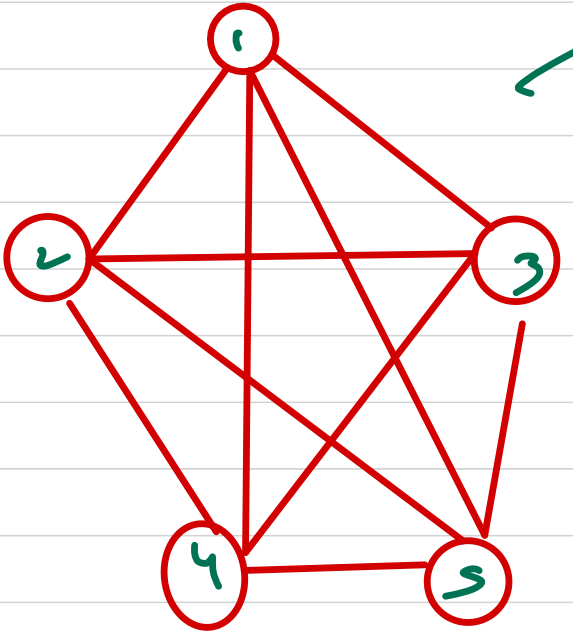
whether there is a path from u to v.




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



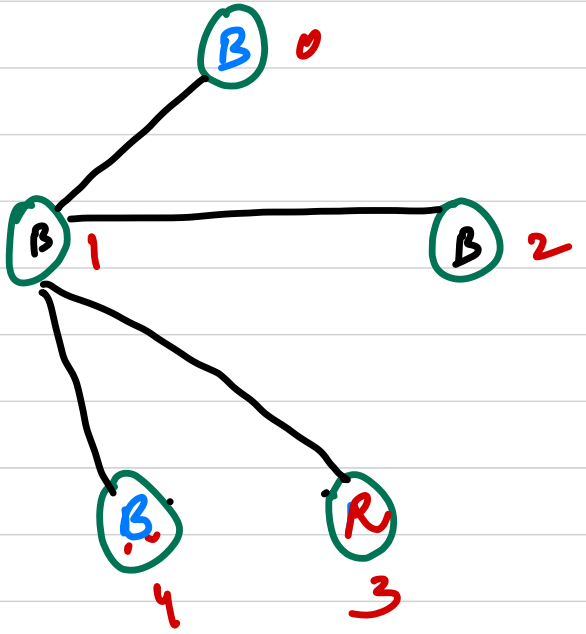
→ complete graph

1, 5

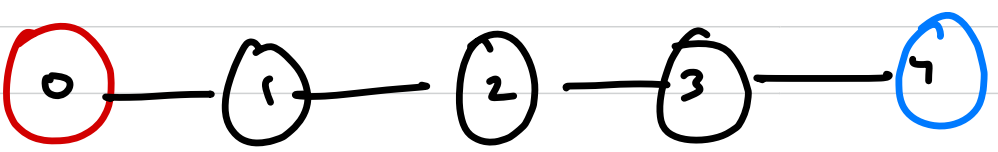
O(v!)

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.

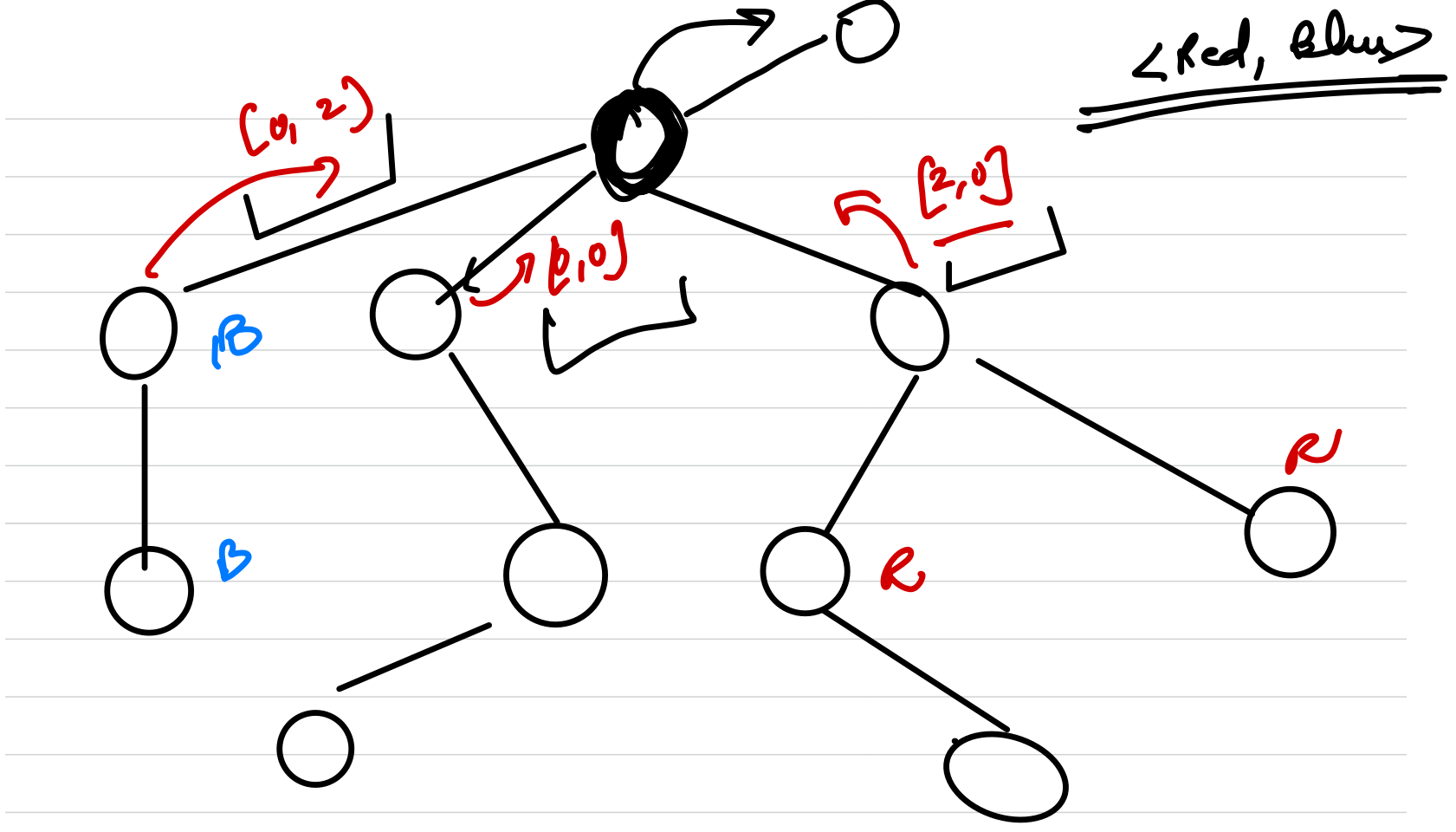
$V \leq 3 \times 10^5$



1 ano

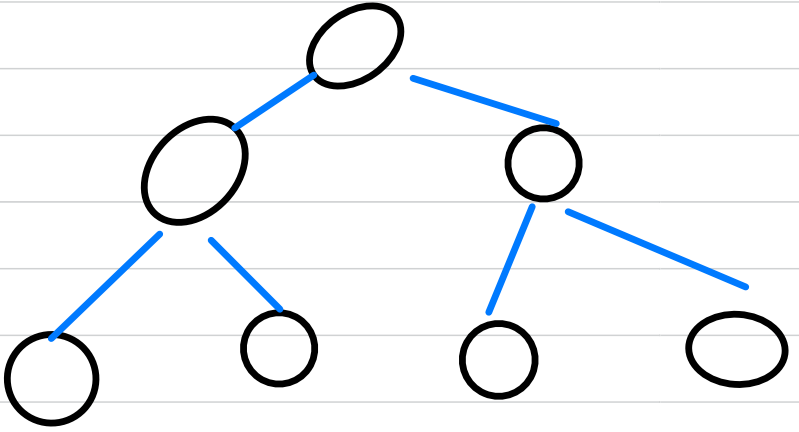


4


<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.

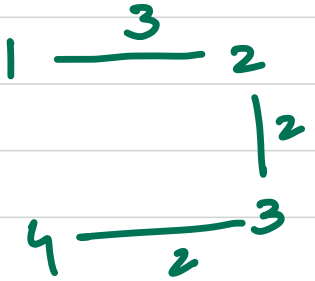
$$f(n+1) = f(n) + 1$$
$$= (n-1) + 1 \Rightarrow (n+1) - 1$$

$f(n) = n-1$ hence proved.

→ Holi (Spoj)

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

18 ← belta



1 ⊂ 2
4 ⊂ 3

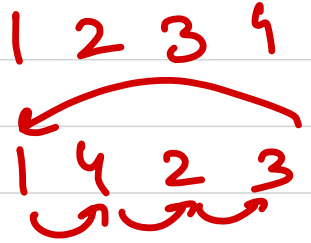
1 → 3
2 → 3
4 − 2
3 − 2

10

Brute force

1. 2. 3. ..... N → We can have N!

permutations



7 — 9

N nodes → (N − 1) edges

1 — 4

Tree (weighted)

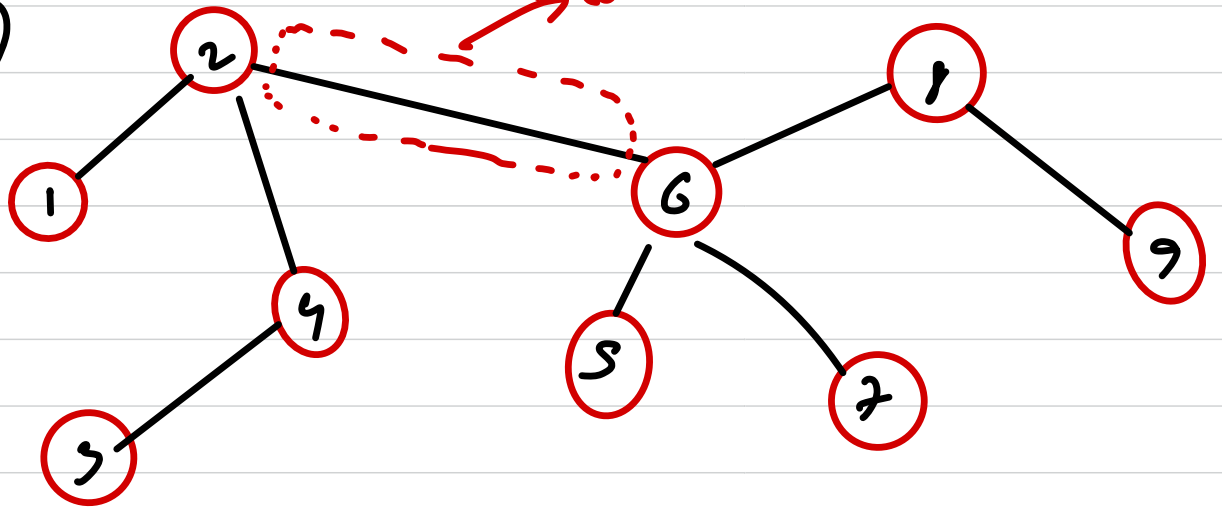
max contribution of each edge

$E \times O(V+E)$

$VE + E^2$

$O(V^4)$

ICE

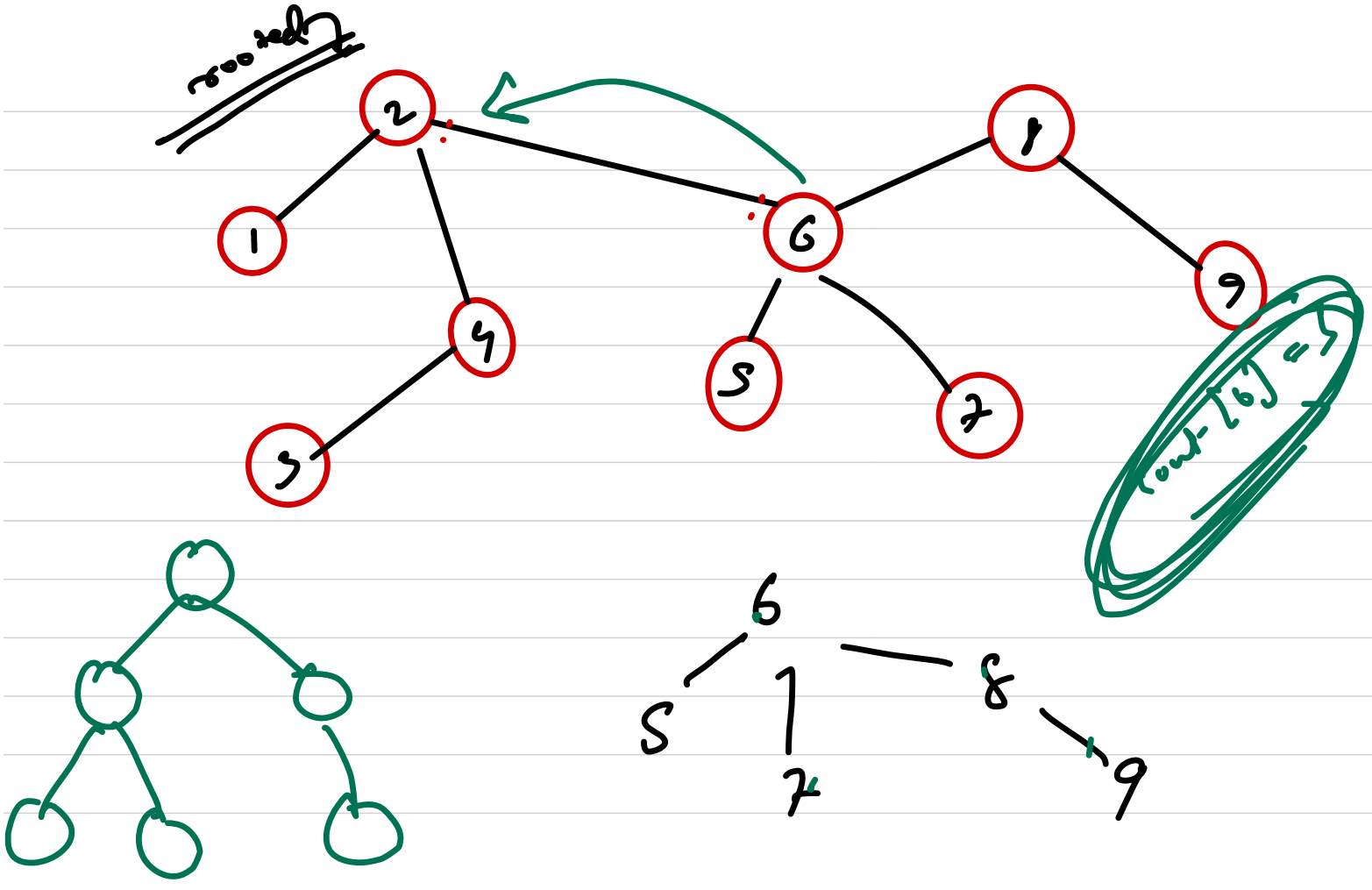


1 — 5 5 — 2

2 — 6 6 — 3 . . . .

For any edge e,

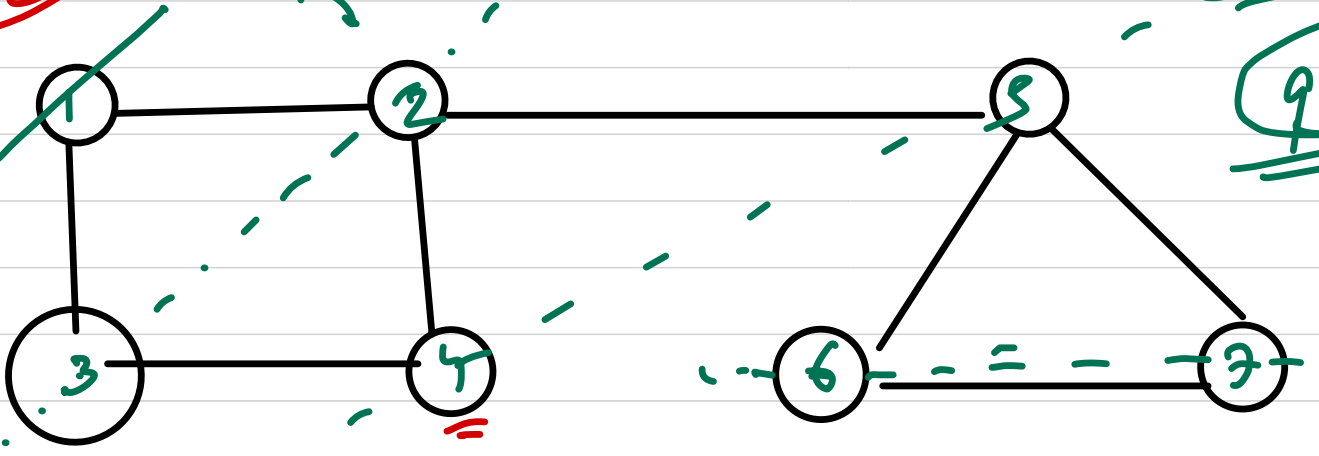
Contribution = $2 \times wt \times \min(s_2 - c_1, s_2 - c_2)$


root node
2
1
4
3
6
5
7
1
9
count[6] = 1
6
5
1
7
8
9

vis

Breadth first Search

FCFS
FIFO
queue



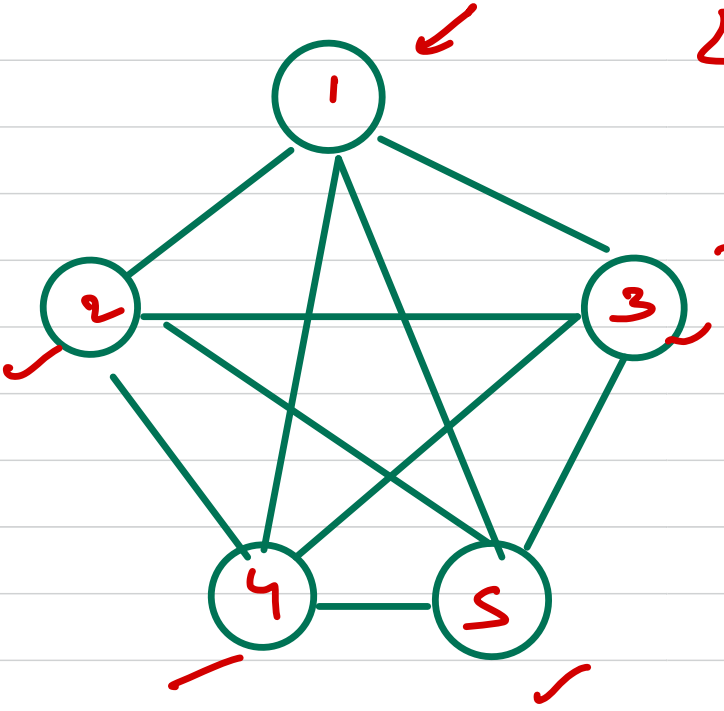
queue

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)



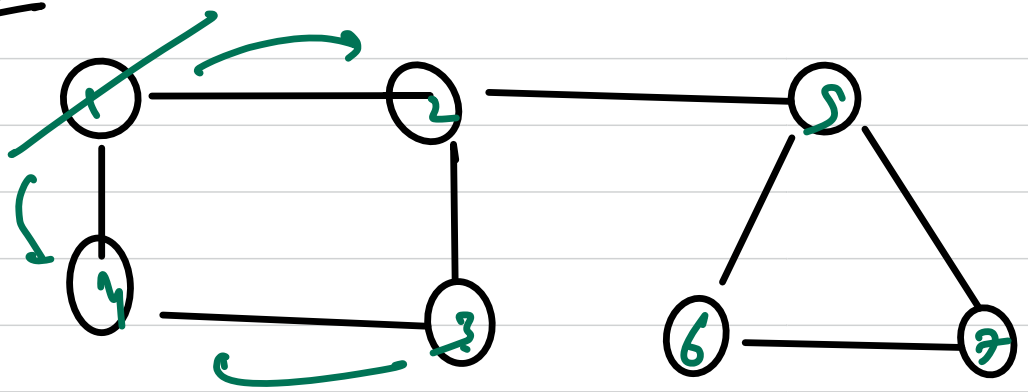
1	2	3	4	5
x	x	x	x	x

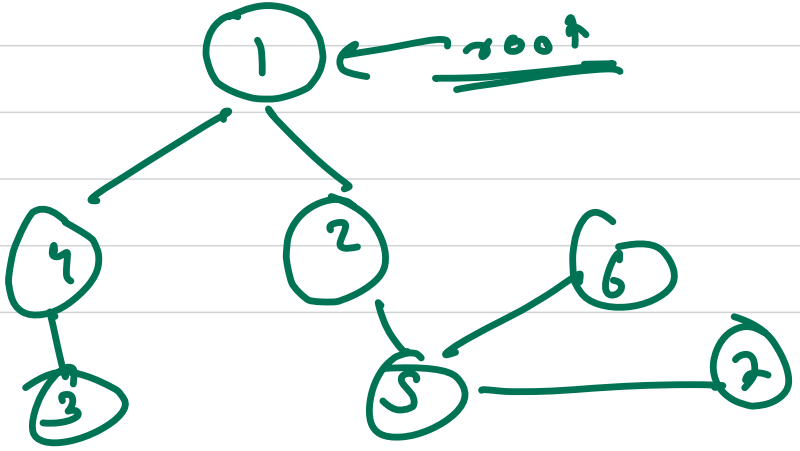
1 2 3 4 5

BFS Tree

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

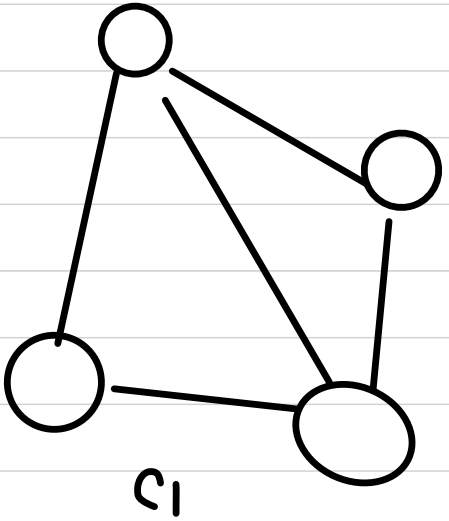
parent [, 1 , 1 , 2 , 5 , 5 , 7]



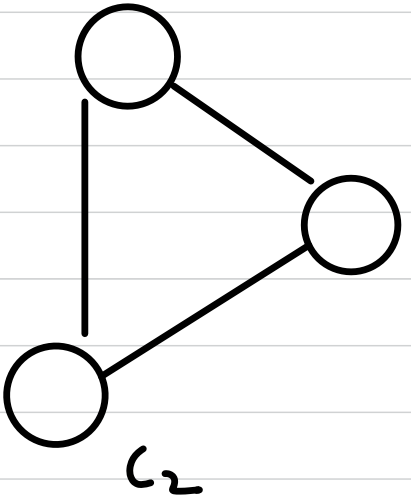


BFS Tree

★ Connected Components



C_1



C_2

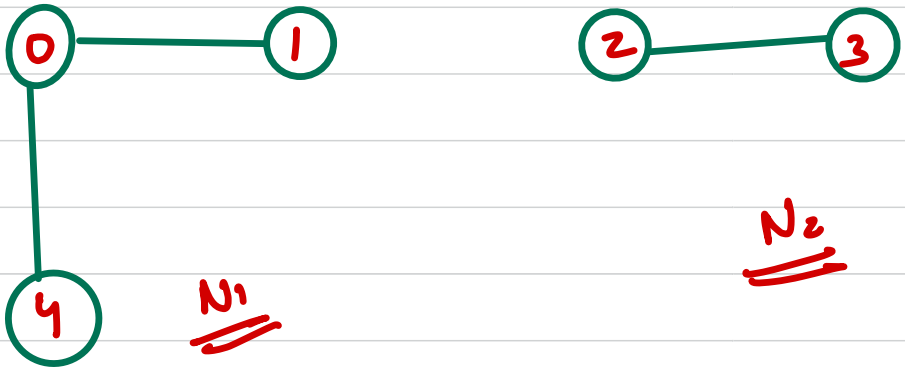
G

Q → Journey to the moon

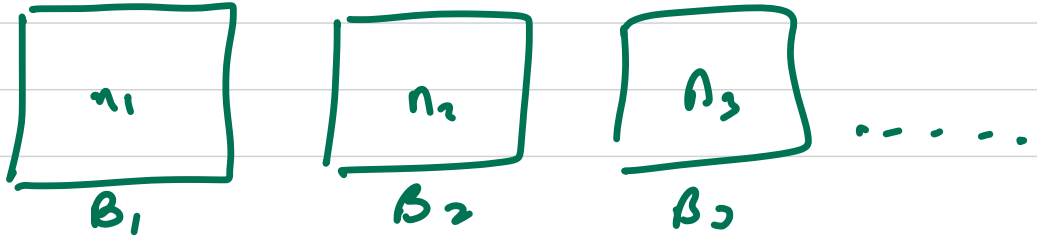
$^nC_2$ $^nC_2$ → Total

total − Σ pairs of same nationality

DFS



# of times of calling 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

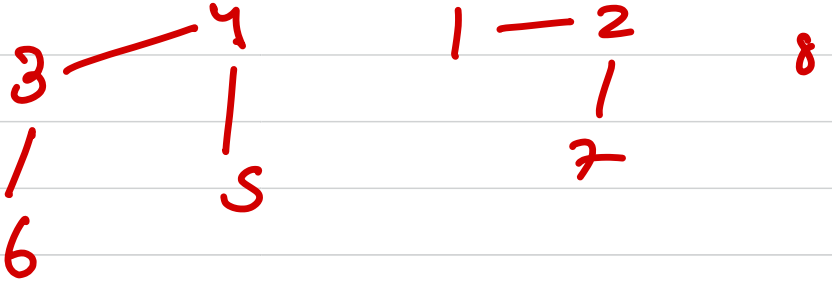
→ connected component ← direct application

index cc

3, 4 (1,2)

4, 5 (2,7)

3, 6



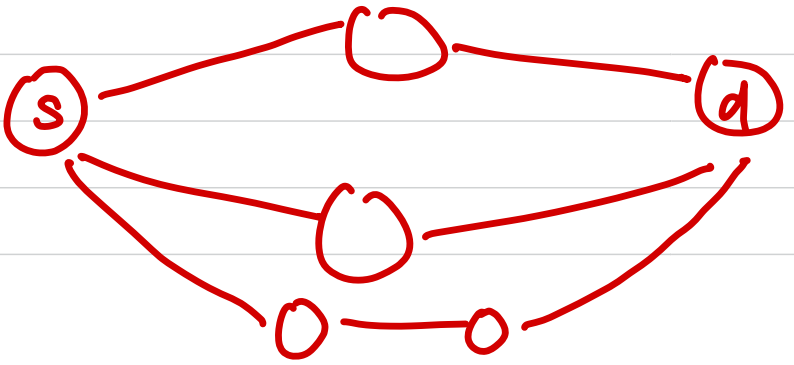
Set → G → [3, 4, 5, 6]

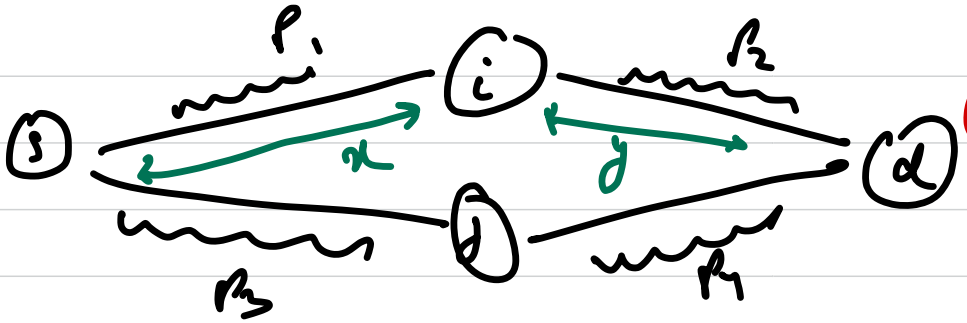
G →	1	2	3	4	5	6	7	8
P →	7	1	4	6	5	3	2	8

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

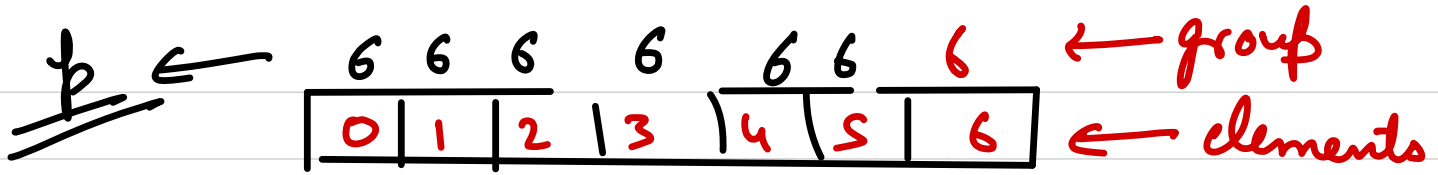
↳ We want to create clusters

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)

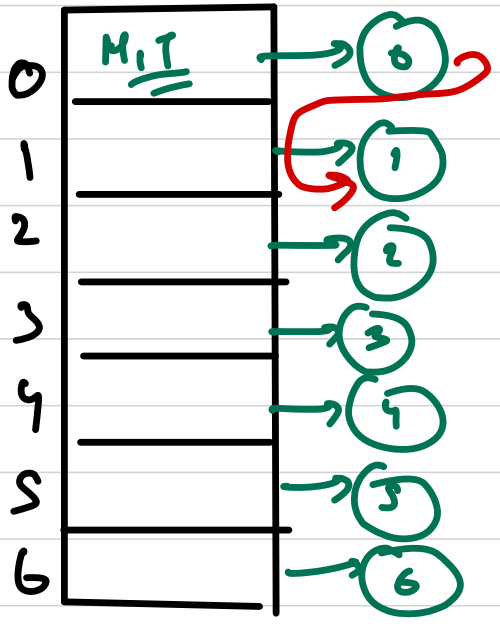
union(2,0)

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)

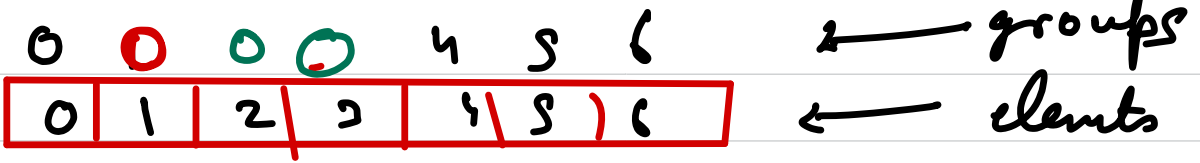
O(1)

// 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

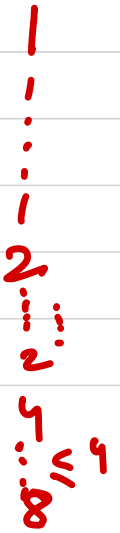
union by size

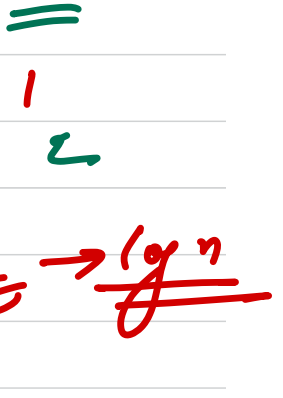


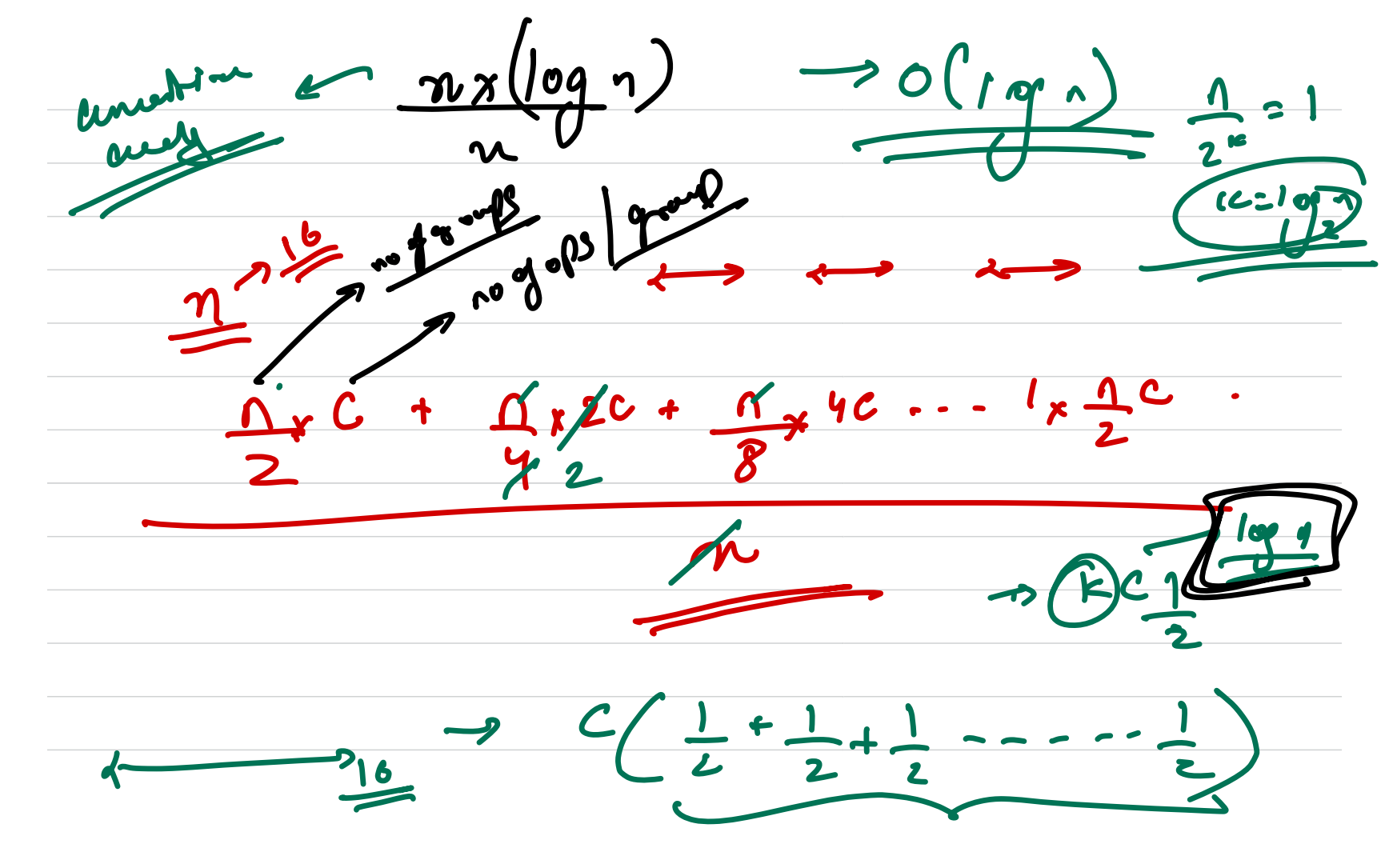
union (0,1)

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

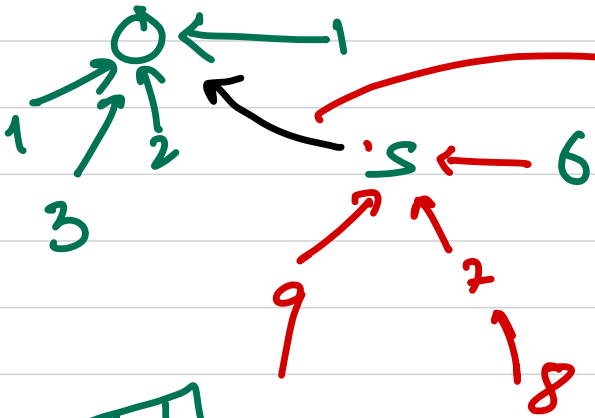
→ get → O(1)

→ min → O(log n) ✓

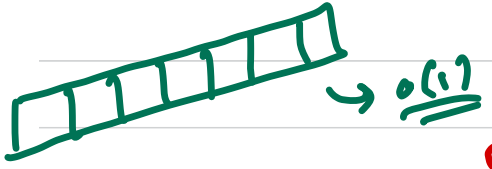
Rank → level

union by rank

same size

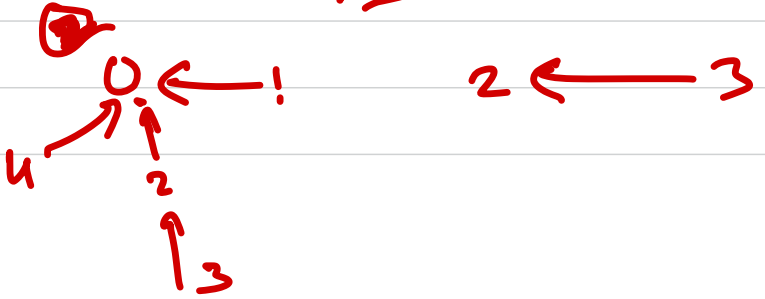


where is the new link added ?!



an image of almost same size

8



4 5 6

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.

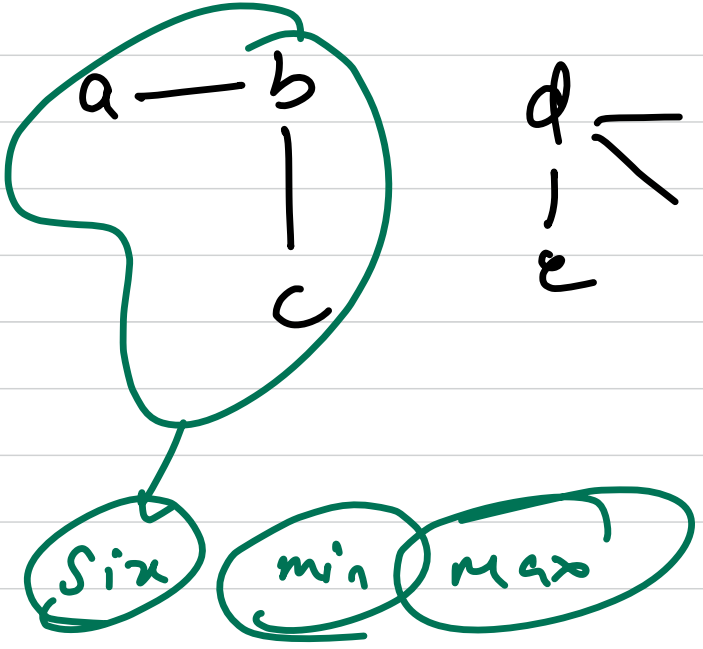
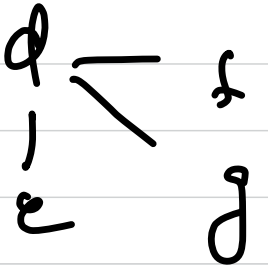
$\log(2^{16}) \rightarrow 16$

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

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

1

connected component

→ $O(V+E)$

↳ $O(\log^* V)$ ←