The Graph Algorithms Course

Pre-Requisite

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

Outcome Expectation

- Course will be starting from very Basics
  but we will cover adv CP topics
- Beneficial with Interview perspective too.

(21 classes)

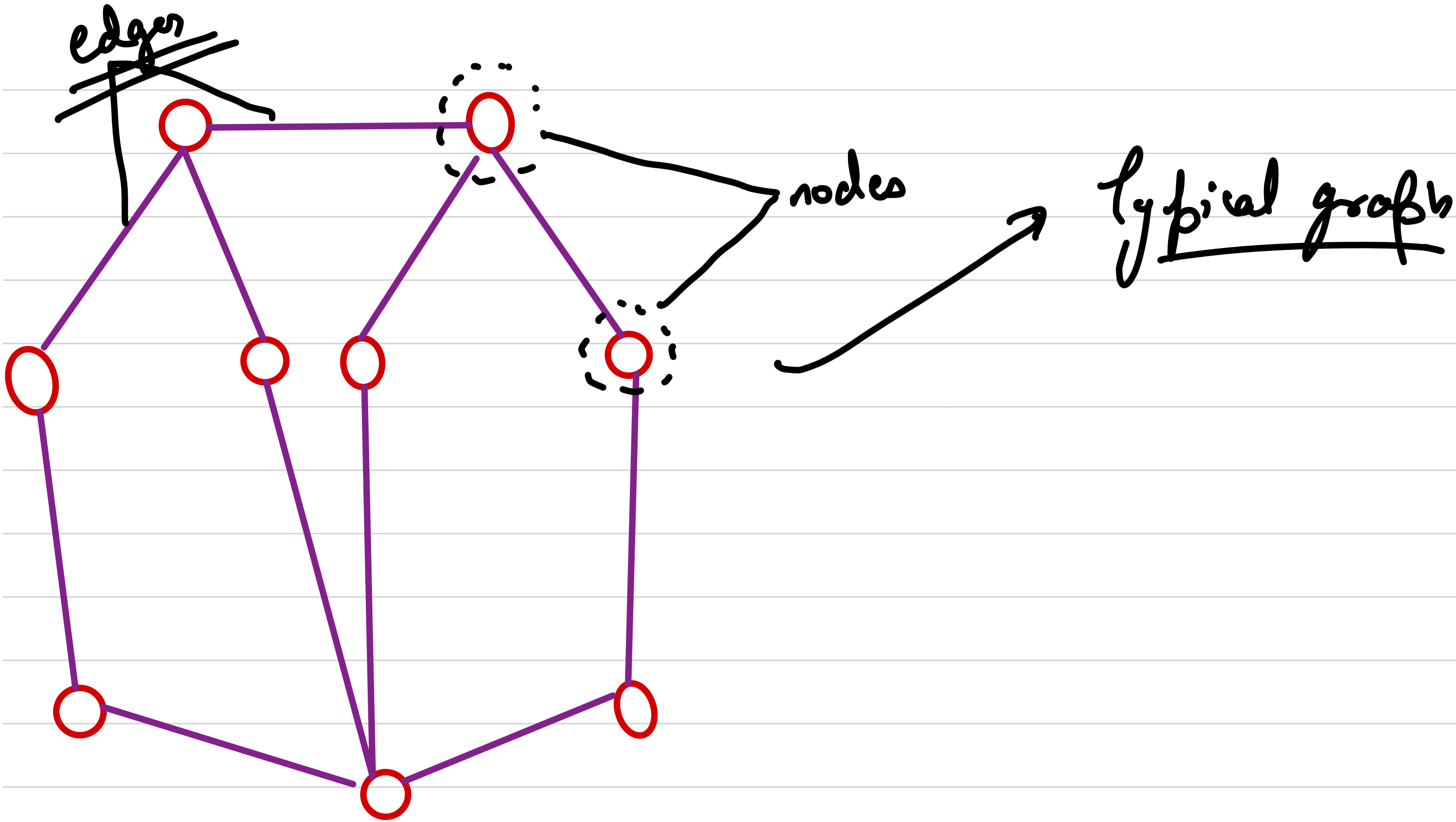
Topics

- Basic graphs & terminologies
- Traversal techniques BFS / DFS
- DP on graph (DP on DAG)
- Matrix BFS / DFS, 0-1 BFS etc
- Trees (n-ary), DP on trees
- Shortest path, MST
- DSU
- SCC, Bridges / articulation point

Graphs

Qs What is a graph ??

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.


edges
nodes
typical graph

Applications of graph

Biology → PPI graph
→ Metabolic network graph

Electrical → Circuit org graph

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

Formal definition of graph

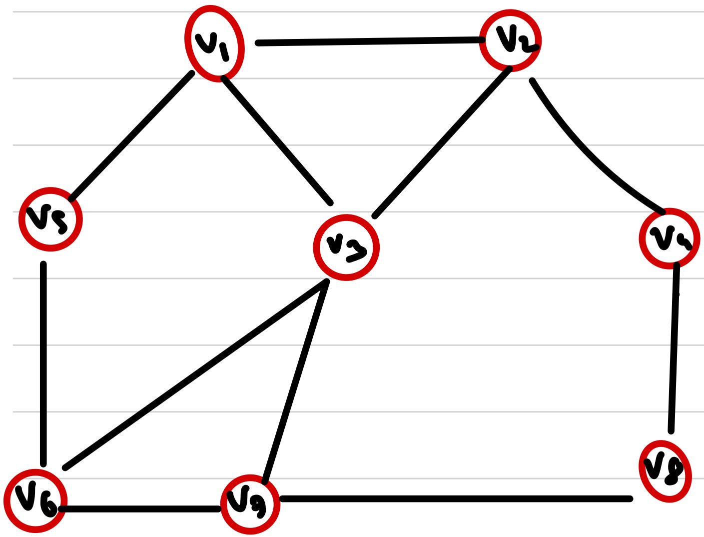
set of vertices

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

Edge set

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

unordered set



$G = (V, E)$

mathematical notation of graph

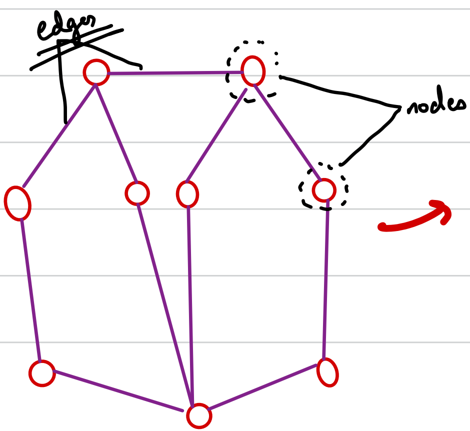
A graph G is an ordered /unordered pair of a set V vertices & E, edges.

Types of graph

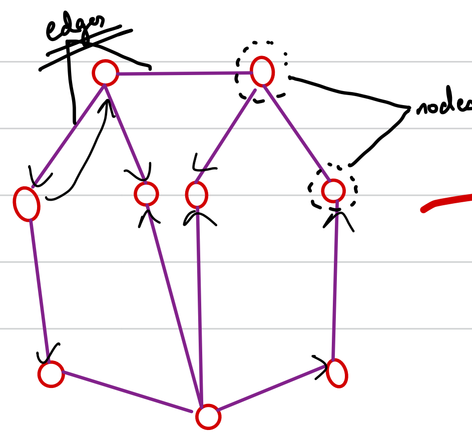
Based On direction

→ Directed → water flow, road network,

→ Undirected → facebook



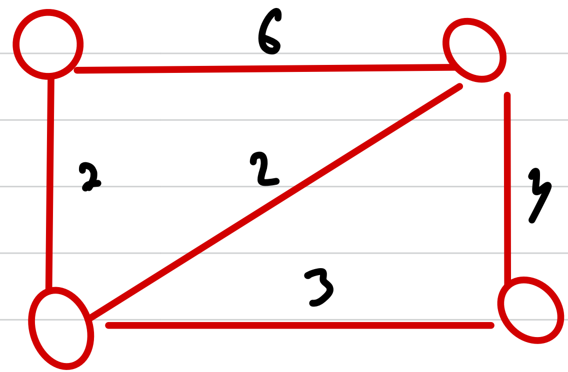
→ undirected



→ directed

Based on edge property

1. Weighted
2. Unweighted



Based on edge density

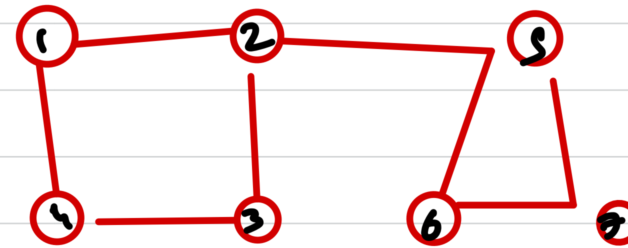
1) Sparse

2) Dense

Representing graphs as a data structure

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

adjacency matrix



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

$$A = \begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 7 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

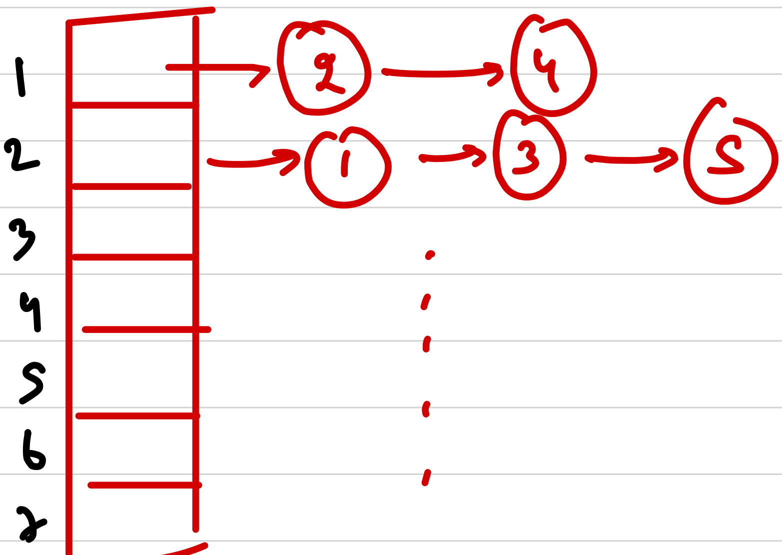
$\longrightarrow \underline{O(V^2)}$

$\underline{N \times V}$

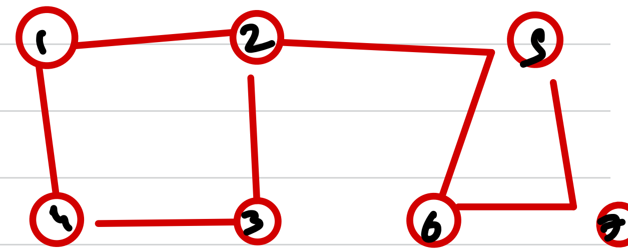
adjacency List

{v, w*}

Array of LL



Bucket array



$O(V+E)$ → space

adjacency map/set

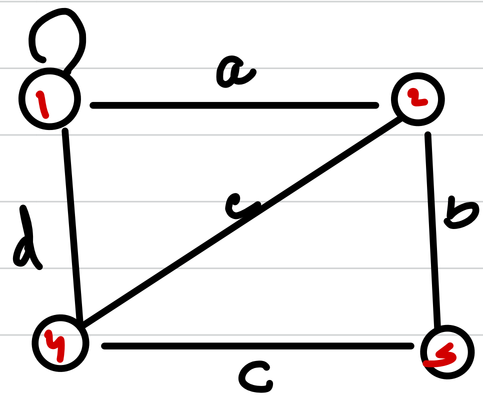
array of hashmap

Incidence Matrix

5 min

$$M = \begin{array}{c} \\ \\ \\ \\ \\ \end{array}\begin{array}{ccccc|c} a & b & c & d & e & \\ \hline 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 2 \\ 0 & 1 & 1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 1 & 1 & 4 \end{array}$$

$(V \times E)$



col → edges
rows → vertex

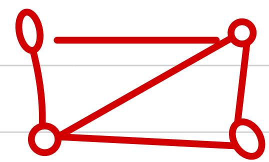
$$M_{ij} = \begin{cases} 1 & \text{if the } i^{th} \text{ vertex belongs to the } j^{th} \text{ edge} \\ 0 & \text{else} \end{cases}$$

col sum → 2

row sum → degree

Q. What is a degree??

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



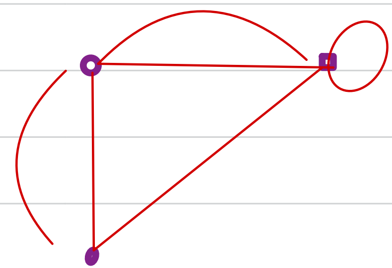
directed graph → Indegree/outdegree

outdegree → no. of outgoing edges

indegree → no. of incoming edges

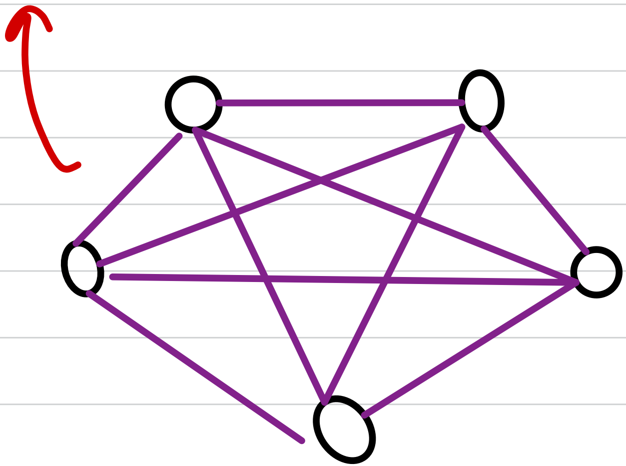
Multi graph → an undirected graph with multiple edges btw vertices & self loops allowed.

2 multi graph



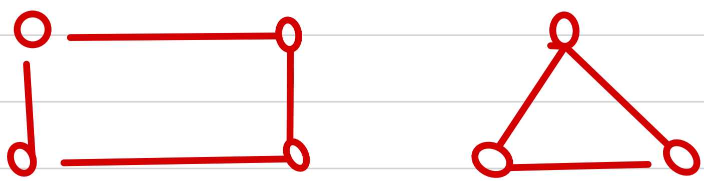
Simple graph → An undirected graph in which both multiple edges & loops are not allowed.

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



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

Disconnected graph → atleast 2 vertices do not have a path to every other vertex.



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 such that

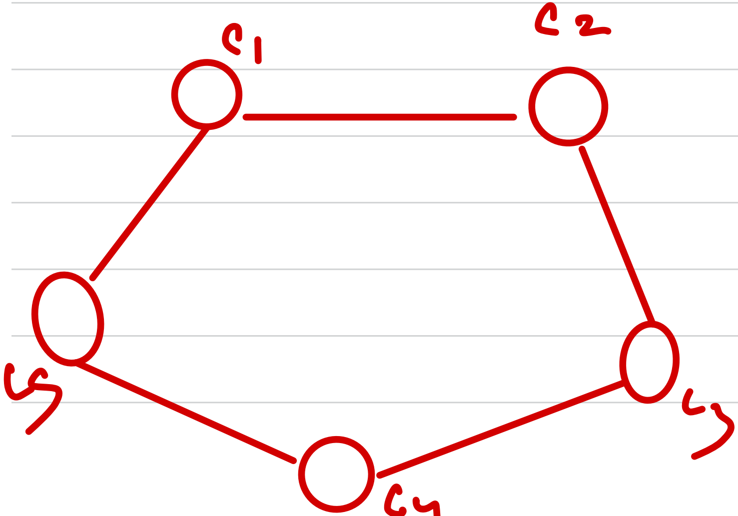
$$V = \{v_1, v_2, v_3, v_4\}$$

edge set of the graph is

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

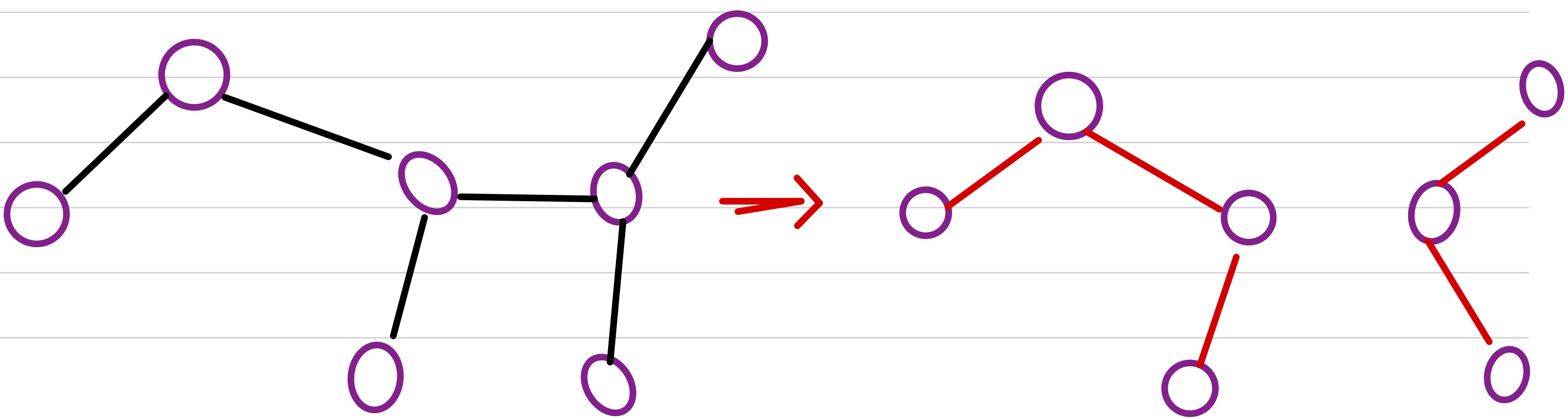
Cycle → A cycle C_n is a graph whose vertices can be arranged in a cyclic sequence such that edge set is

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

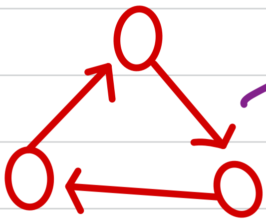


TREE → Tree is a connected graph with no cycles

forest → If we remove an edge from tree we get a forest which is 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 traversal

→ Depth first

→ Breadth first

Depth first search

TC

$O(V+E)$

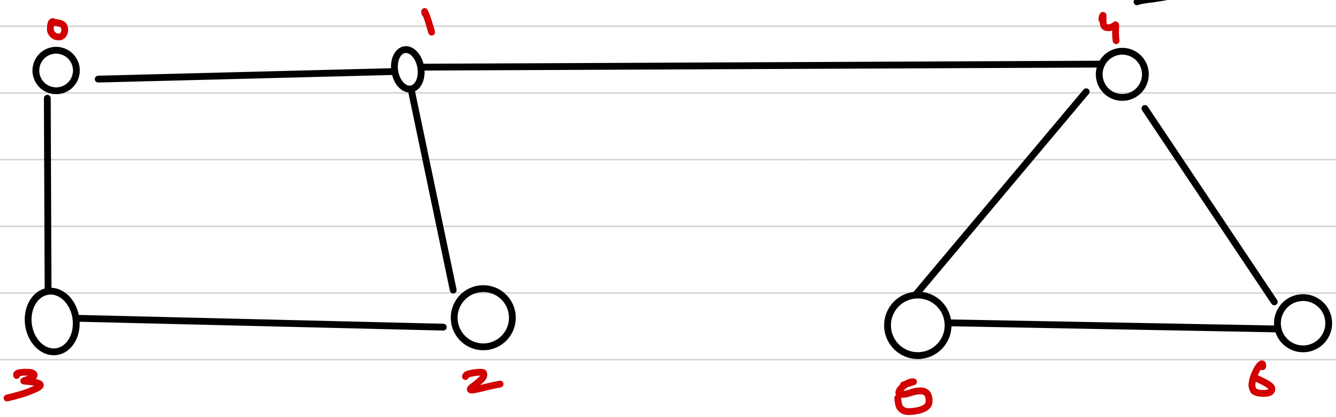
*** Motivation problem:**

Q1 Given a graph calc all paths between two vertices

OR

Q2 Given a graph check whether there is a path between 2 vertices.

Last in → stack



is there a path from 0 - 5

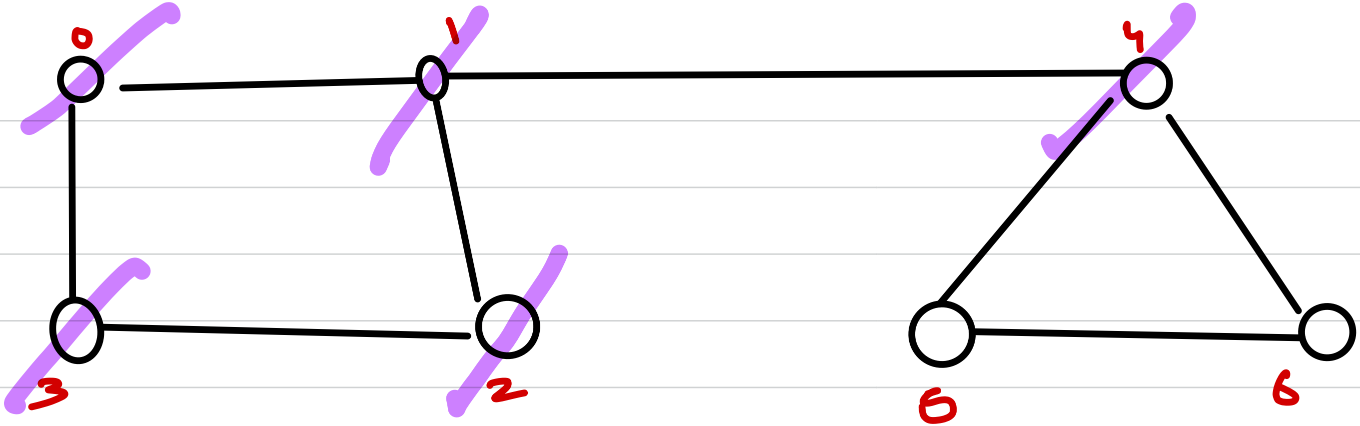
What is the most easiest query for this ??

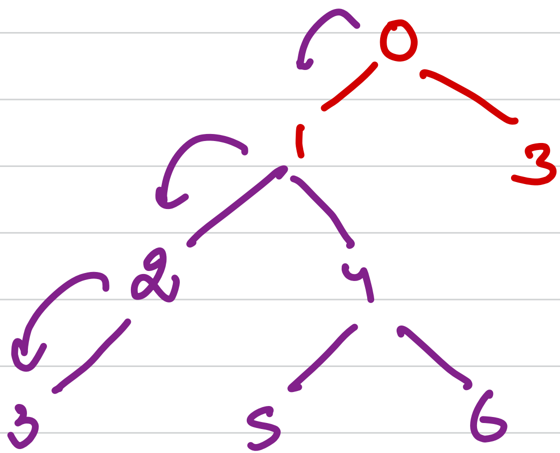
→ neighbours

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

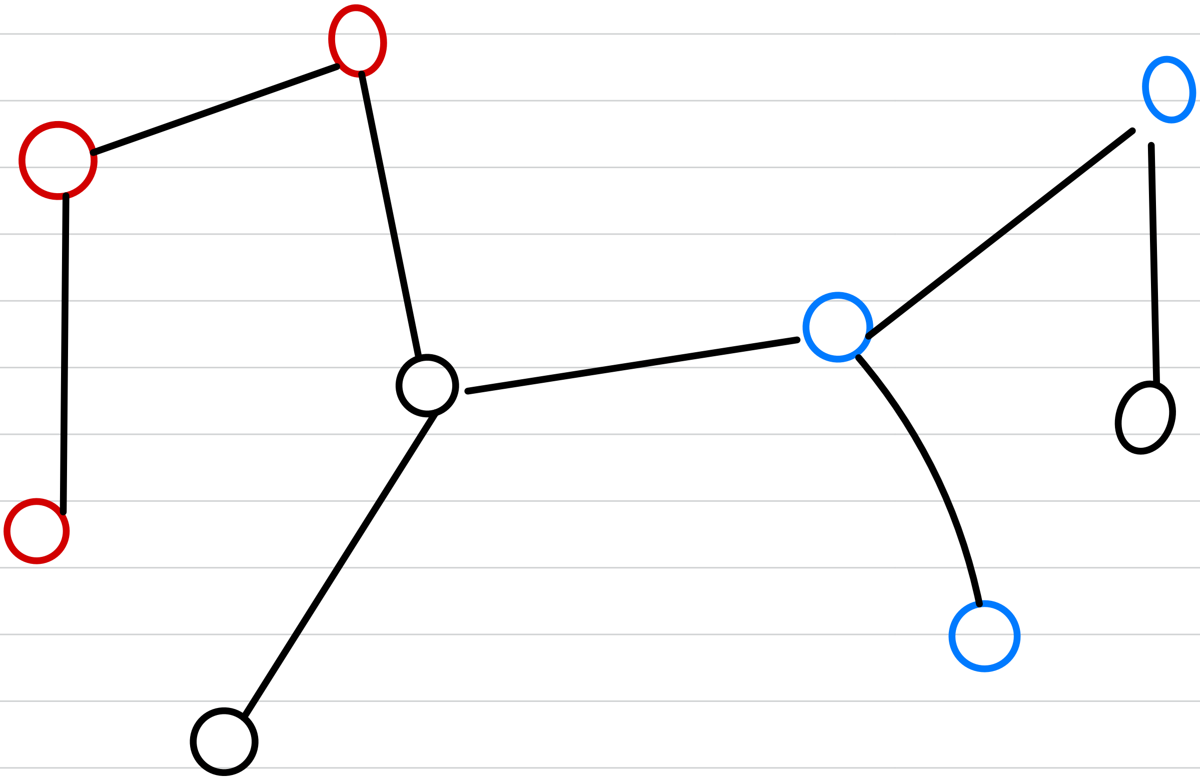
$\uparrow$
where there is a path from u to v.

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


0
1
4
3
2
5
6


0
1
3
2
4
3
5
6

→ total red
total blue



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

PMI → $f(n)$ ↳ no. of edges for n nodes

$f(1) \rightarrow 0$

$f(2) \rightarrow 1$

To proove → $f(n+1)$

we assume function works for $f(n) \to n-1$

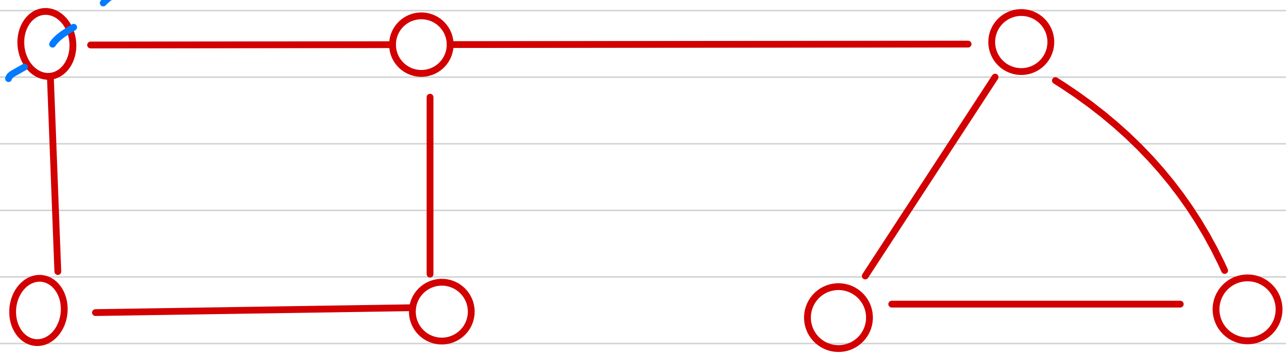
Total no. of edges → no. of edges required for $(n+1)^{th}$ node + $(n-1)$ edges

Every node that will be added to a tree requires only 1 node.

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

equivalent to level order traversal

Breadth First Search

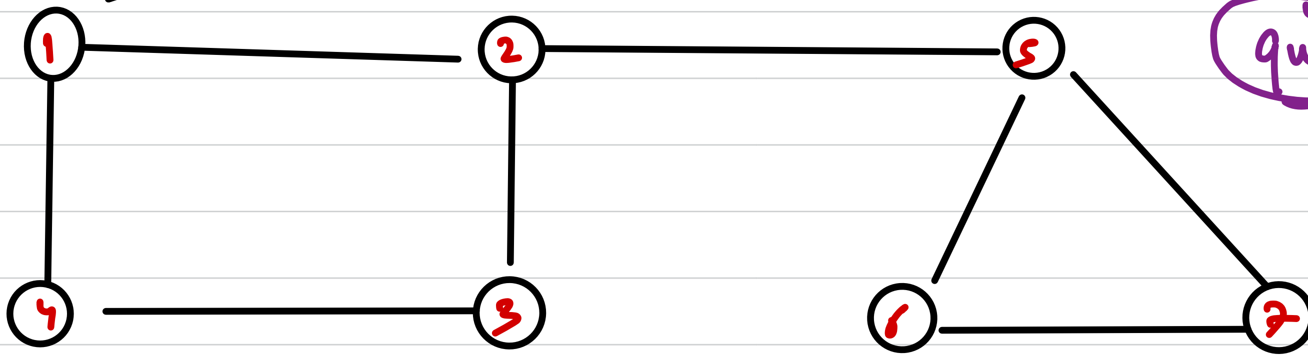


Say we have an arbitrary src node then all the nodes with same shortest (by edges) distance from src node are said to be on the same level.

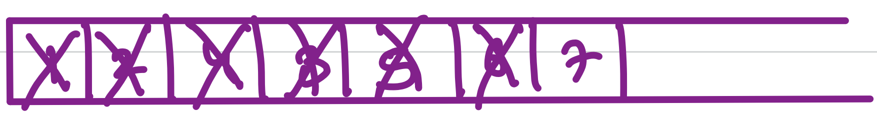
start

first in first out

queue



1, 2, 4, 3, 5, 6, 7 → Breadth first traversal

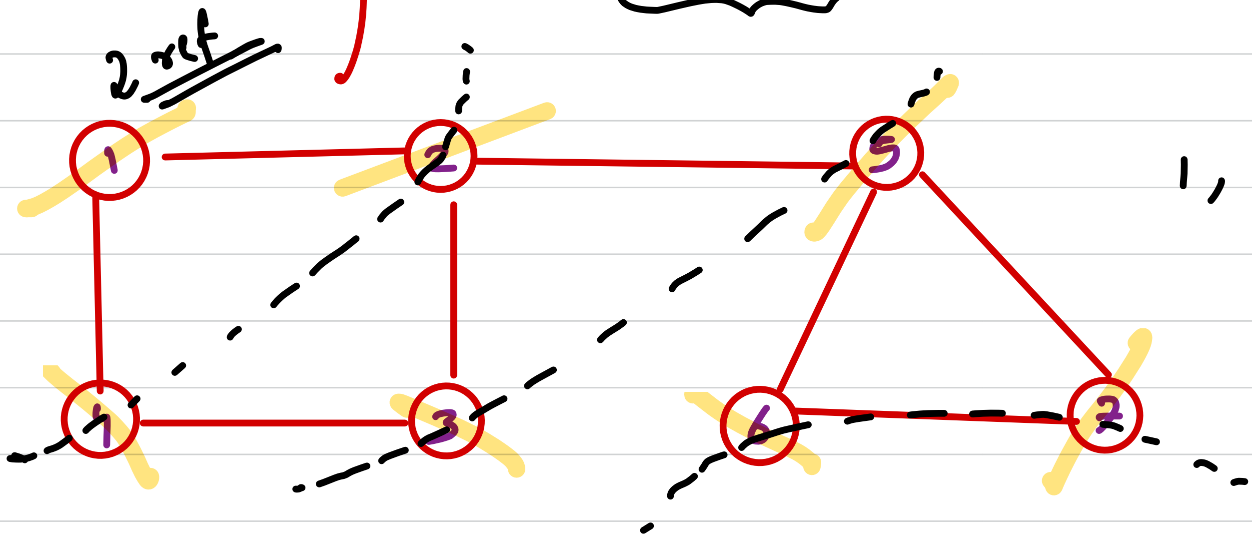


→ queue

Breadth First Search

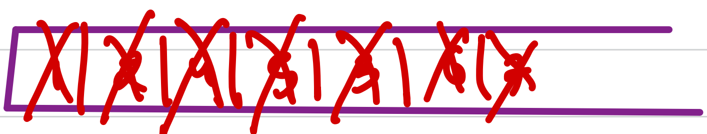
FCFS

unweighted



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

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



queue

Time → O(V+E)

Space → O(V)

So in breadth first search, whenever we travel from any node → any node then it always considers shortest path.

Connected Components → DFS / BFS

2 CC

CC

Disconnected graph

vis = { }

iterate on all possible vertices

```
for (i = 1; i <= 7; i++)
    if (not vis[i])
        DFS(i)
```

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

Journey to the moon

pair of ast id < a_1, a_2 > then a_1, a_2 are from same country

Choose → 2

Say, total no. of ast $= n$

$${}^nC_2 \rightarrow \frac{n!}{2!(n-2)!} \Rightarrow \frac{n(n-1)}{2}$$

? 2

$\langle a_1 \ a_2 \rangle$

$\langle a_2 \ a_3 \rangle$

$\langle a_3 \ a_4 \rangle$

$\langle a_5 \ a_6 \rangle$

~~graph~~ G

$a_1 \quad a_2$

$a_3 \quad a_4$

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

$a_5 \ a_6 \rightarrow 1$

$${}^6C_2 \rightarrow \frac{{}^{3}\cancel{6}\times 5}{\cancel{2}} \Rightarrow 15$$

$$15 - 6 - 1$$

$$\Rightarrow 8$$

Permutation Swaps

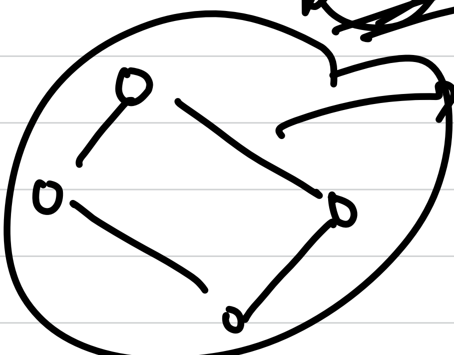
idx	1	2	3	4
A →	1	3	2	4
B →	1	4	2	3

[11, 21, 31]
1 2 3

1, 2
2, 3

dfs

(3, 4) ← index



Those indices are in the same connected component idx 1 2 3 1 3

A → 1 4 3 2 5

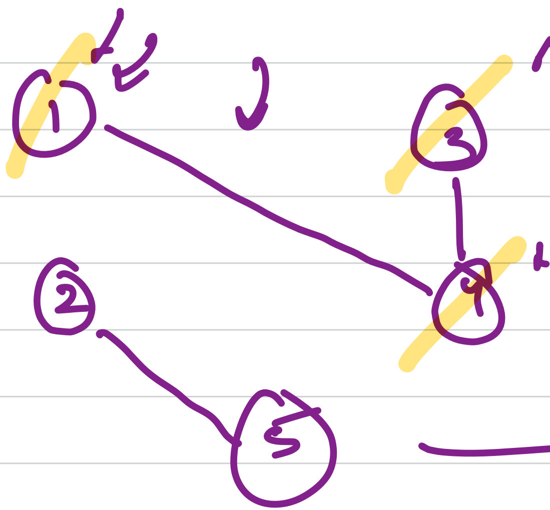
B → 1 5 2 3 4

(2, 5)

(1, 4)

(1, 3)

CC of index



nodes

p → [1, 2, 3]

q → [1, 3, 2]

p → [4, 5]

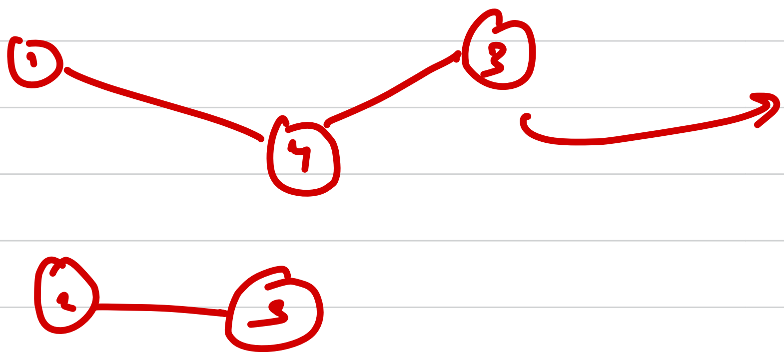
q → [5, 4]

idx	1	2	3	4	5
A	1	5	4	3	2
B	1	4	2	3	5

(2, 5)

(3, 4)

(1, 4)

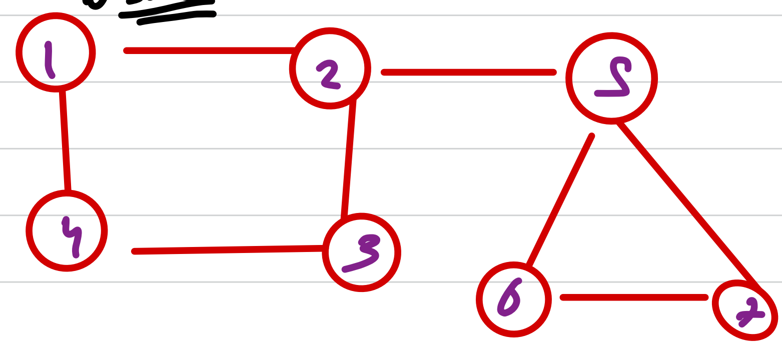


$p \rightarrow [1, 3, 4]$

$q \rightarrow [1, \cancel{3}\ 2, \cancel{4}\ 3]$

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

2 src



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

output

S+C=1

BFS

Bfs traverse the nodes based on shortest path

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

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

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

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

$a+b = x$

$b = x - a$

Rotten Oranges

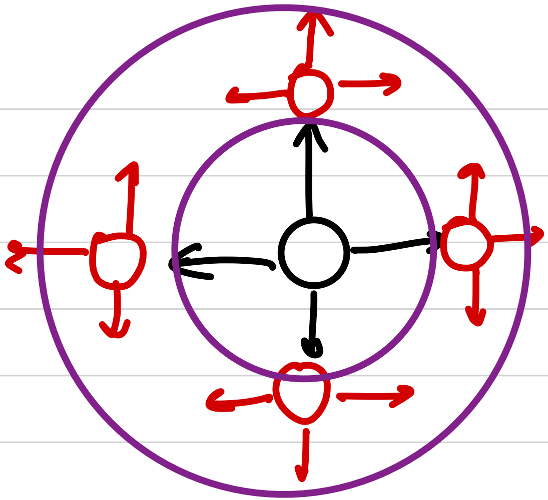
0	2	~~1~~ 2	~~1~~ 2	2
2	~~1~~ 2	~~1~~ 2	0	0
0	2 ~~1~~	0	~~1~~ 2	2
0	0	2	2	2
0	0	0	0	0

2D array

m×n

-1 → when no other orange can rot a orange

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



BFS

queue

vis

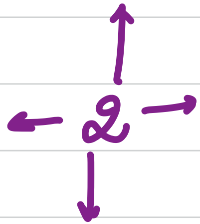
(i, j)

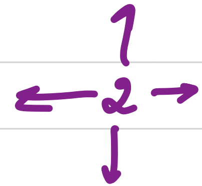
(i+1, j)	(i-1, j)	(i, j-1)	(i, j+1)

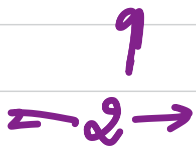
all this will be fresh if this is a fresh orange

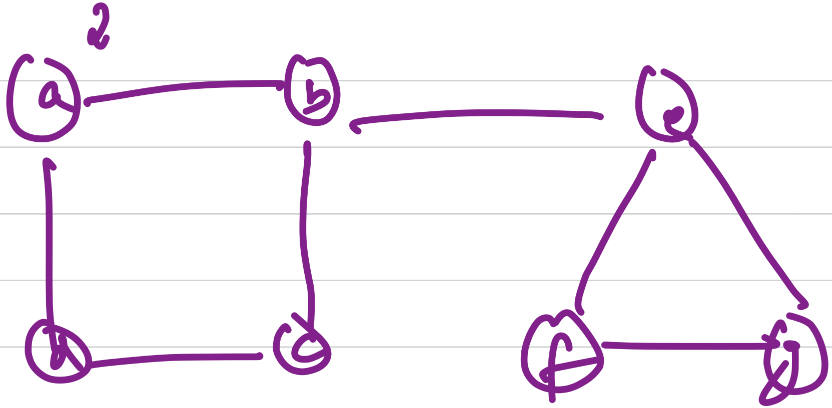
Multi Source BFS

$O(nm)$


2

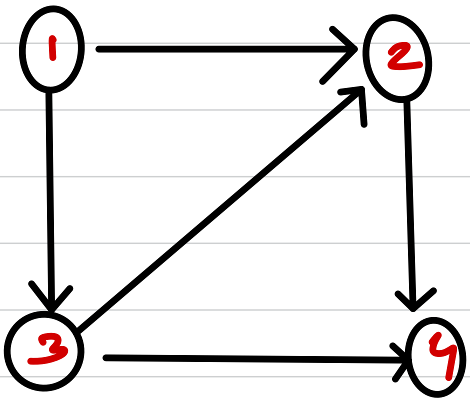

1
2


9
2



~~a~~ | ~~b~~ | ~~d~~ | ~~c~~ | ~~e~~ | f | g

Longest Path   Dynamic Prog.



→ The longest directed path can start from any node.

1D DP

$f(v)$ returns the length of the longest path that starts with v.

$$1 + \max \begin{cases} f(n_1) \\ f(n_2) \\ f(n_3) \\ \vdots \\ f(n_m) \end{cases}$$

Base: leaf $\rightarrow$ 0

where $n_1\ n_2\ n_3 \ldots\ldots n_m$ are the neighbours of v

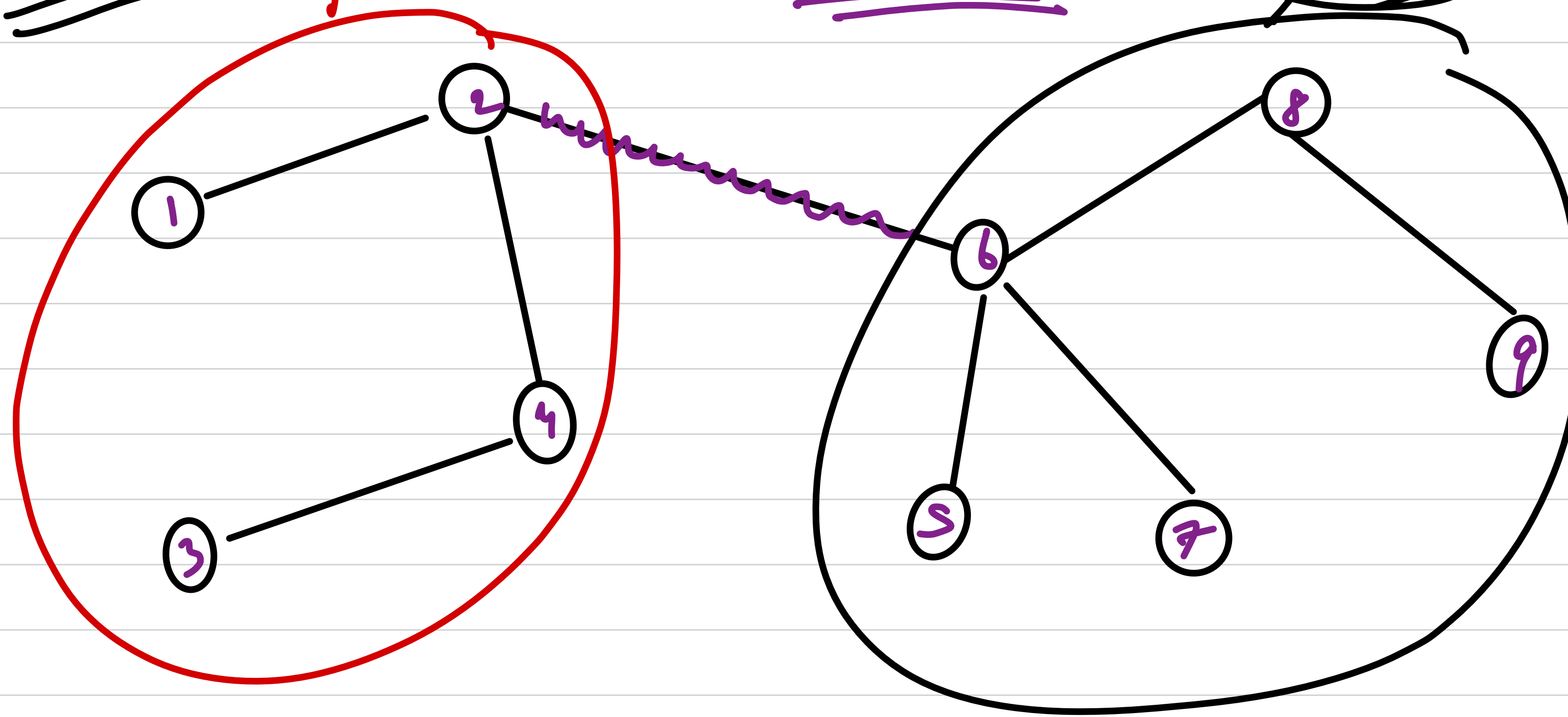
ans $\max(f(1), f(2), f(3) \ldots\ldots f(n))$

DFS

m

HOLI SPOT

n-m

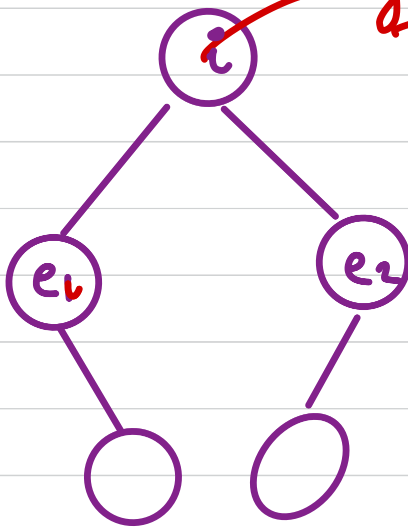


$ax(n-m, m) \times wt$

general director

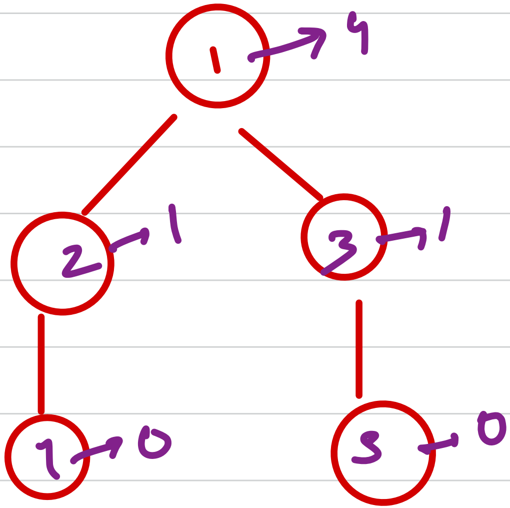
~~1D dp~~

memoize

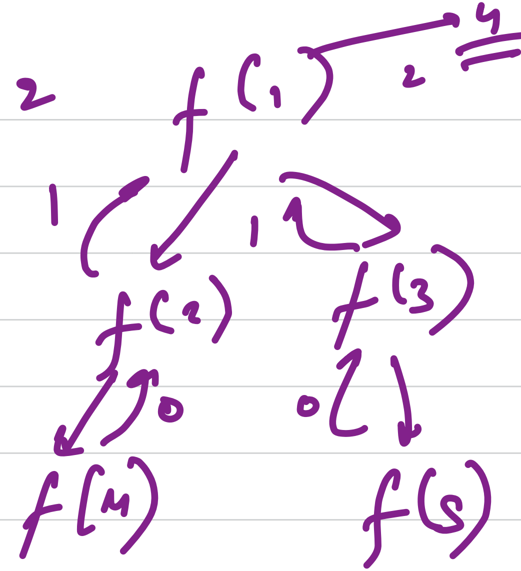


$$f(i) = \sum \left(1 + f(e_i)\right)$$

no. of subordinates under employee i



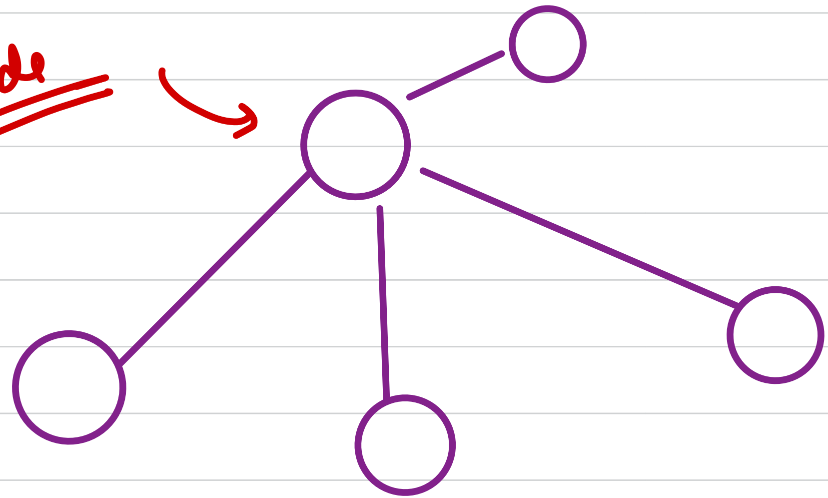
1
2
3
4
5
4
1
1
0
0



f(1)
f(2)
f(3)
f(4)
f(5)
4
2
1
1
0
0
2

Independent Set → Atcoder

node



color → B
→ W

Black → cost
white → no cost

No, two adjacent vertices should be black

$f(node, c)$ → these two parameters can uniquely identify our state

c → constraint

$f(node, 0)$ → # of ways to color subtree rooted at node

as C → 0 then node can be W or B

$f(node, 1)$ → # of ways to color subtree rooted at node and node is under constraint

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

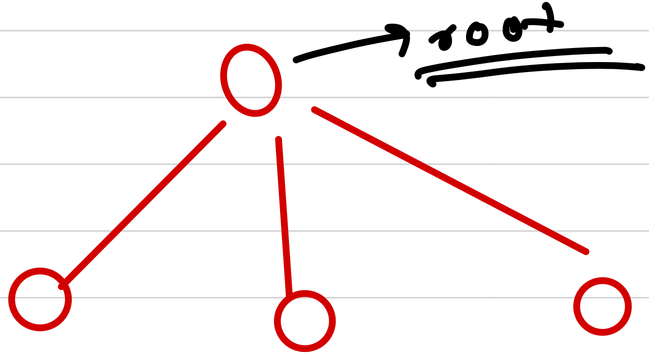
$//$

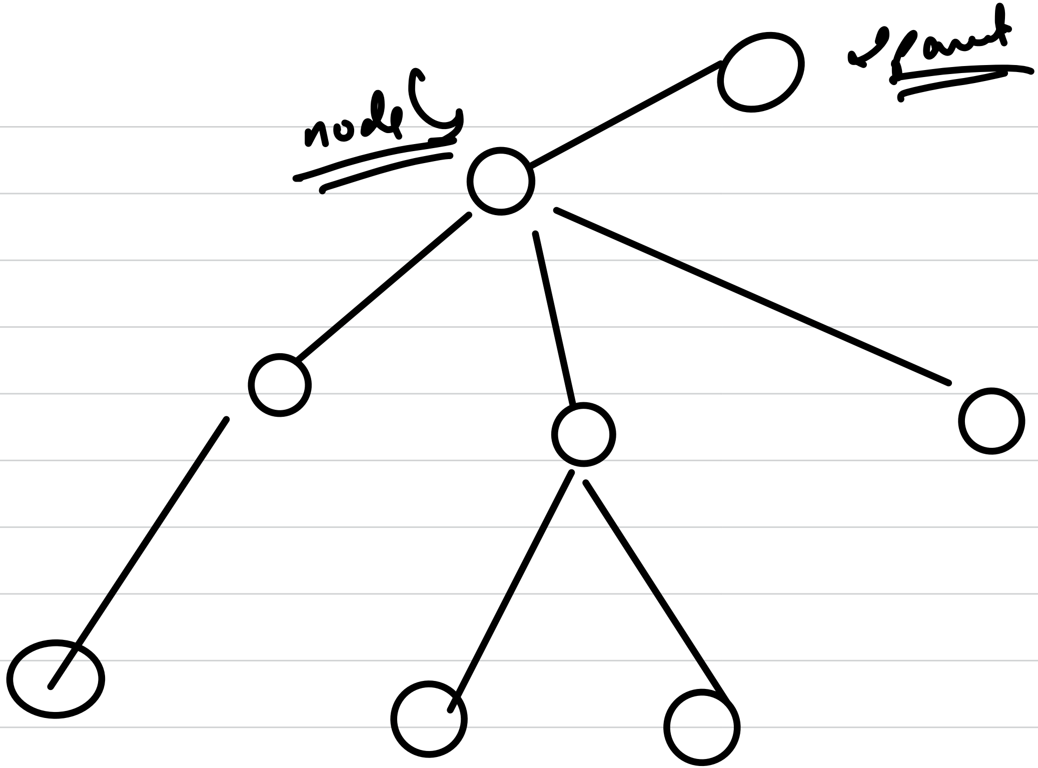
final ans $\rightarrow$ $f(\text{root}, 0)$

Leetcode

Binary Tree Cameras

Brute force

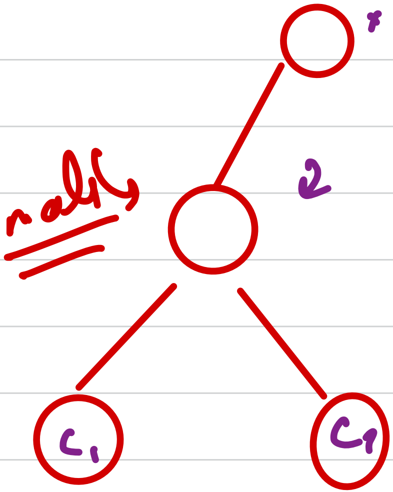



node
parent

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

return the min number of cams to cover the subtree rooted at node.

final ans $\rightarrow$ $\min\left(f(\text{root}, 1, 0), f(\text{root}, 0, 0)\right)$



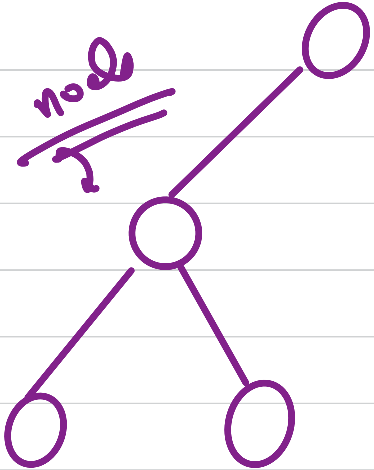
for current node, we decided to have a cam & parent is not have a cam

$$f(node, 1, 0)$$

for your cam

$$f(node,1,0) = 1 + \min\big(f(c_1,1,1), f(c_1,0,1)\big) + \min\big(f(c_2,1,1), f(c_2,0,1)\big)$$

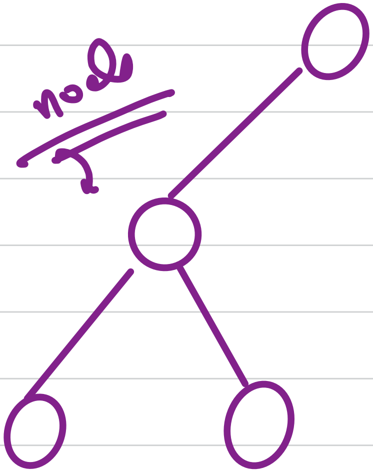
when node is not having cam but parent has it



$$f(node, 0, 1)$$

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

3D dp



$f(node, 0, 0)$

one of the children are bounded to have a **cam.**

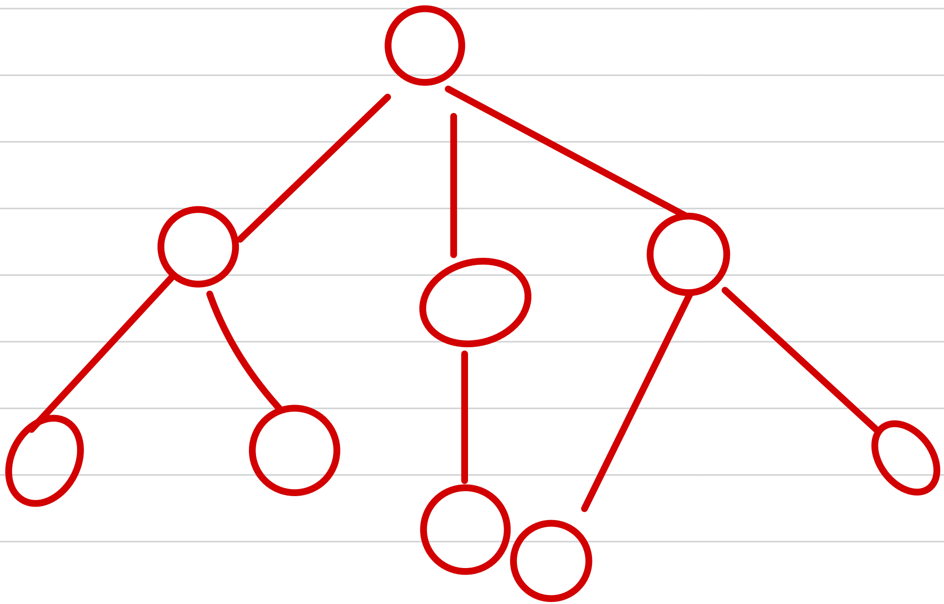
$$f(node,0,0) = \underline{\min}\begin{cases} f(c_1,1,0) + \min\big(f(c_2,1,0),\ f(c_2,0,0)\big) \\ f(c_2,1,0) + \min\big(f(c_1,1,0),\ f(c_1,0,0)\big) \end{cases}$$

and $\leq 10^5$
y

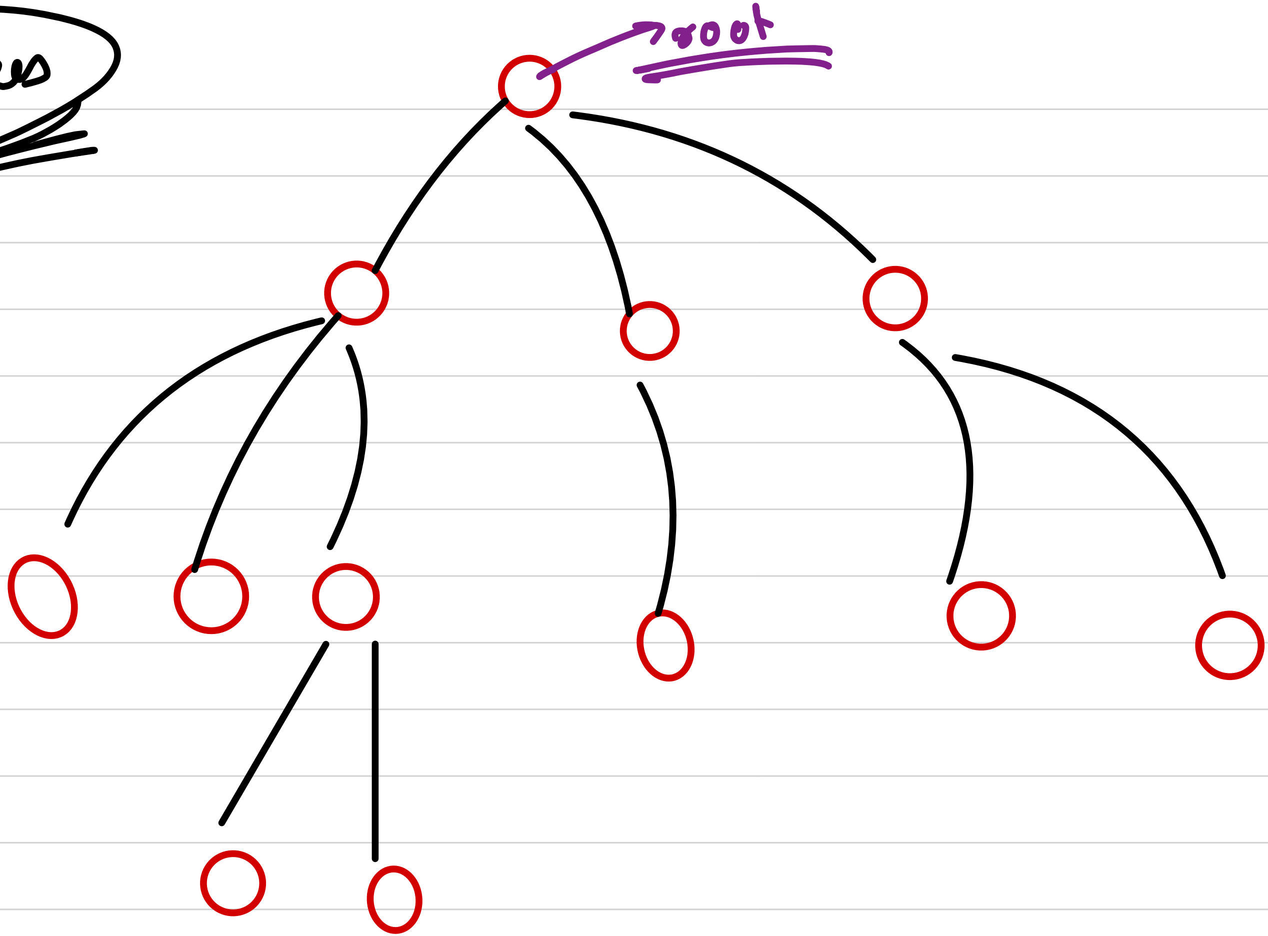
famtree (codechef)

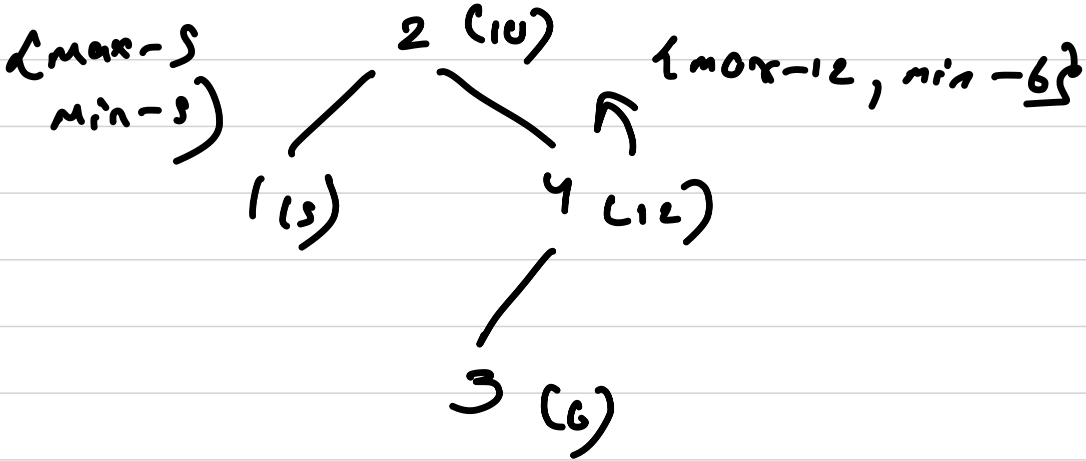
max

min



Dp on trees





10-5

12-10

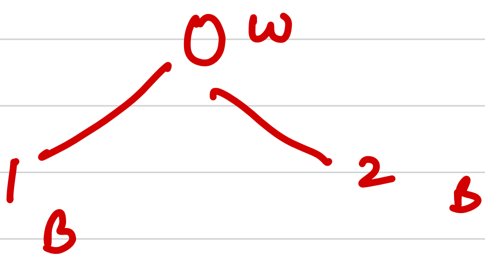
ans → ~~-∞~~ 6

appleman and tree → codeforces

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

$\boxed{0 \le k < n}$

Tree → (n-1)

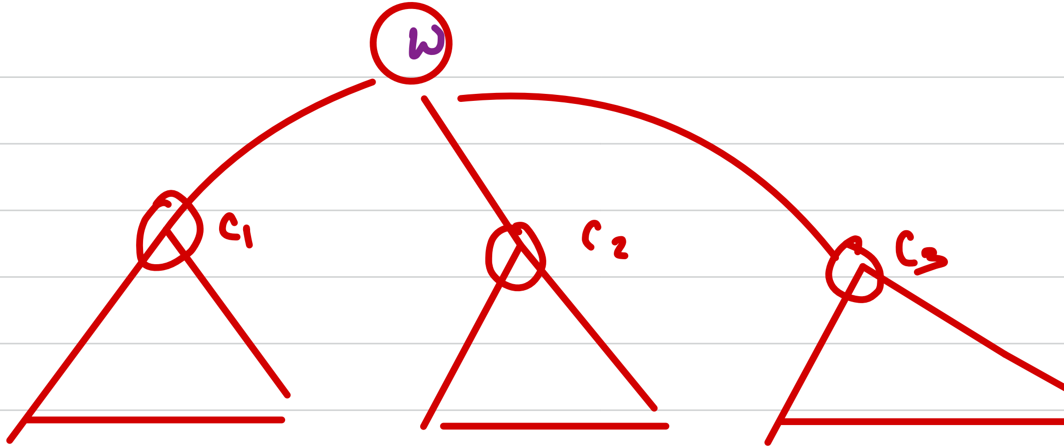


every node will have different choices based on colors of node in subtree

$f(\text{node}, c)$ → count of black nodes

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

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



black

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

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

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

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

Qn You're given a set of inequalities.

Ex ($a<b$ $c>d$ $b<d$ . . . . .)

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

Dependency Resolution

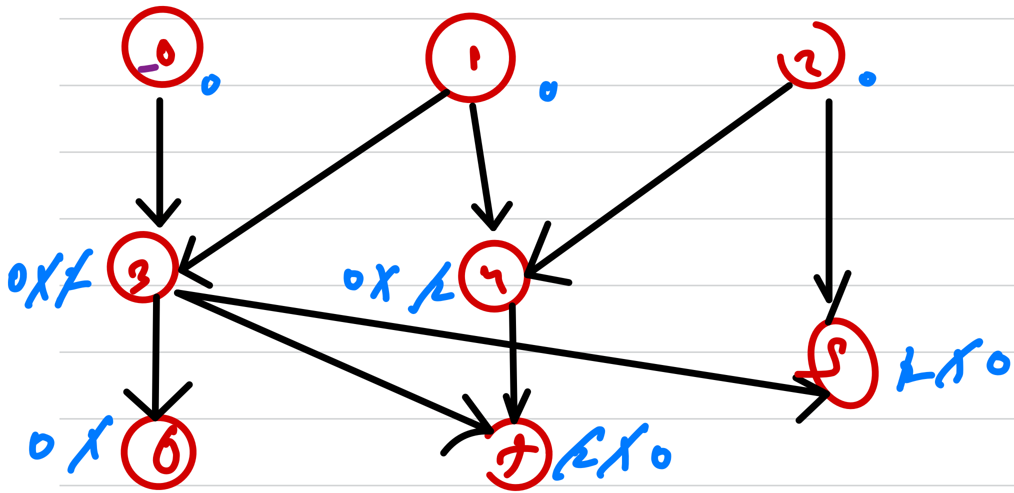
- $a < b$, $c > d$, $b < d$
- packages of a software
- course structure

can be visualised in the form of a graph. → directed graph

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

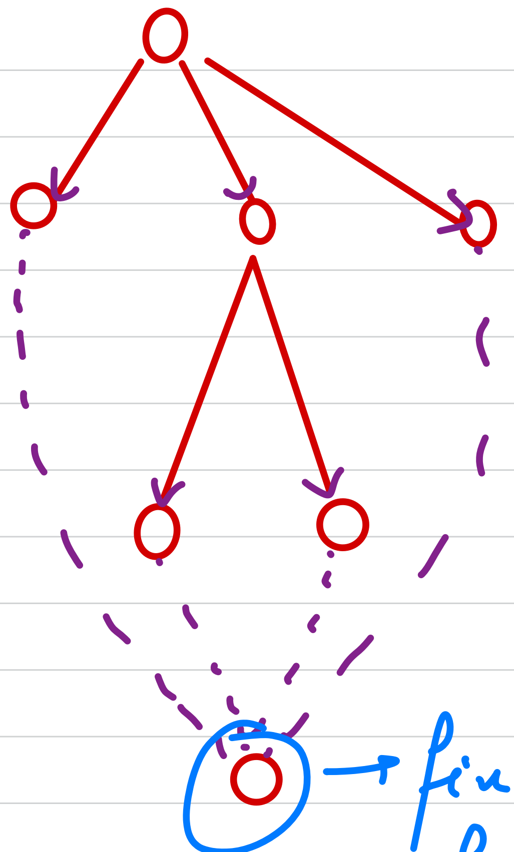
Kahn's algorithm → BFS based topological sort

queue



5, 7, 3, 11, 8, 2, 9, 10

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



2 → 0, 1, 2, 3

fix the value here i.e Node N. 0, 1, 2, 3

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

Q> Given a grid with chars [L, R, U, D] on each cell of grid. You're standing at top left of the grid. From each cell you can go in the direction written on the cell. For ex L → left.

Detect if it is possible to reach the bottom right corner or not.

Dimension → $m \times n$

Expected Space → $O(1)$

$m, n \leq 10^3$

↳ there is only one case for not reaching the bottom right.

↳ if there exist a cycle

cycle detection

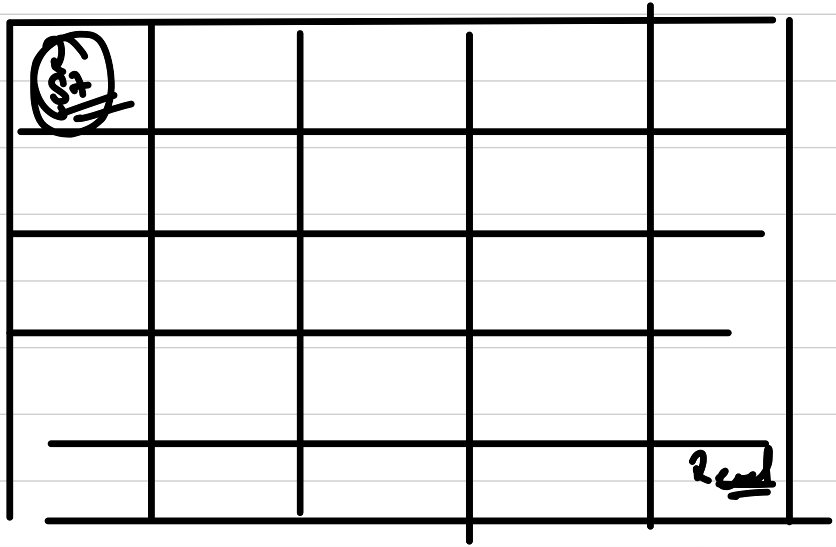
$1 \longrightarrow 1x$

$2 \longrightarrow 2x$

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

Calc the min cost to reach bottom right.

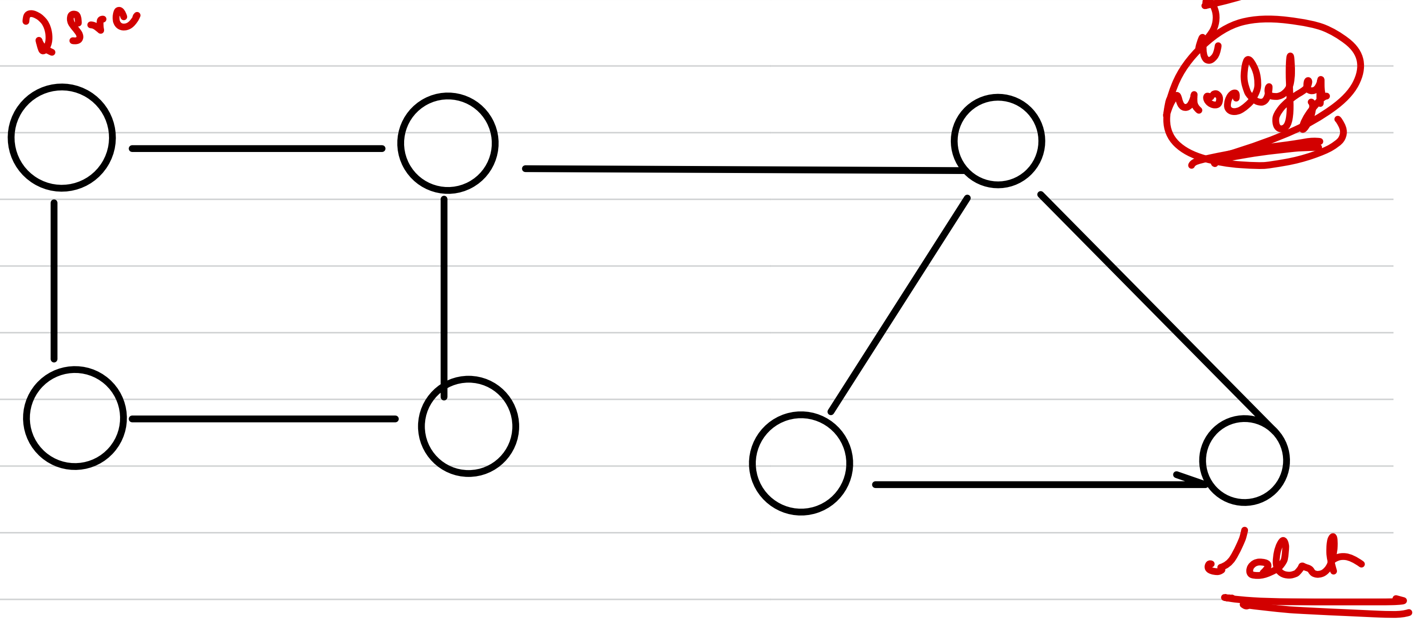
1 unit



On this weighted graph calc the shortest path given a graph & a src and a dest node the shortest path from src → dest

BFS

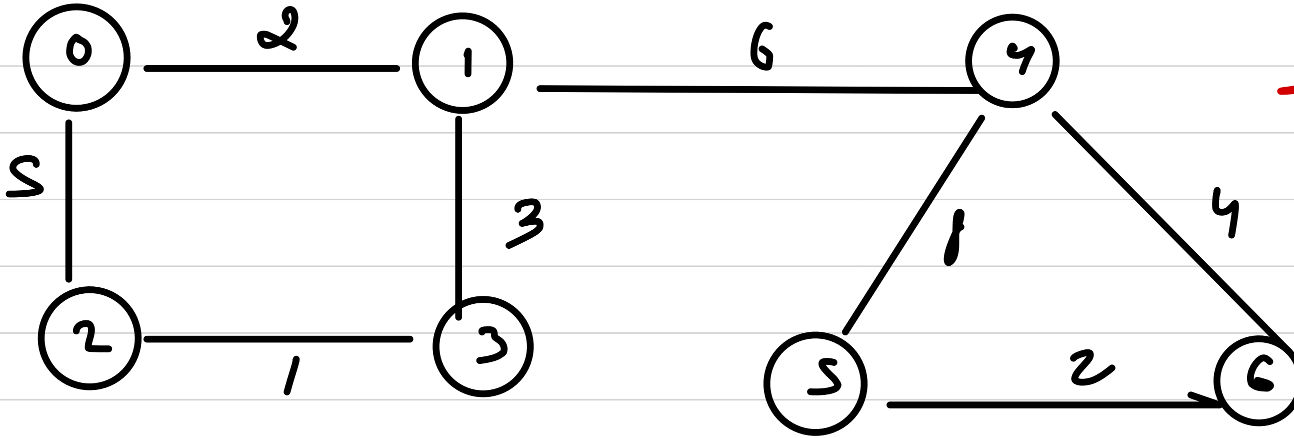
modify



Dijkstra's

Priority Queue

2 src



Valid

pq

node	parent	wt
0	-1	0
1	0	2
2	0	5
3	0	5
4	1	8
5	6	14
6	4	12

Trup

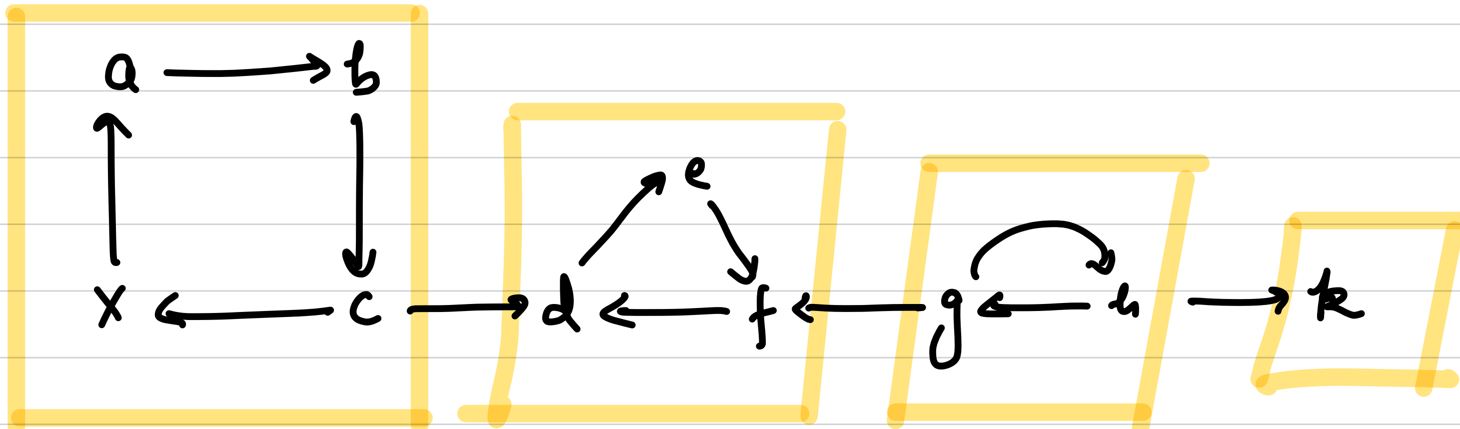
0	2	1	3	4	5	6

Strongly Connected Components

directed graph

We have a path from a to b but not vice-versa

Within a component in a SCC every vertex should have a path to other vertex in a directed graph



How can we find SCC??

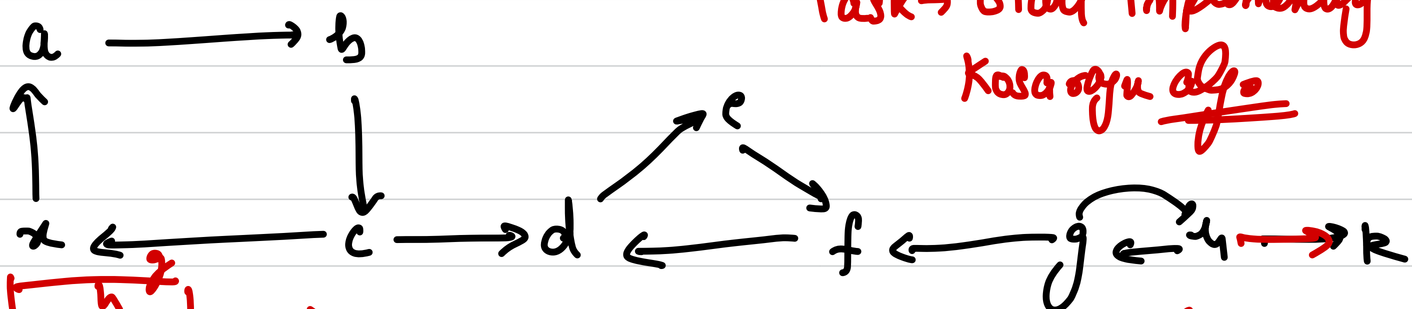
for finding normal cc in an undirected graph,
we used to take help of dfs / BFS

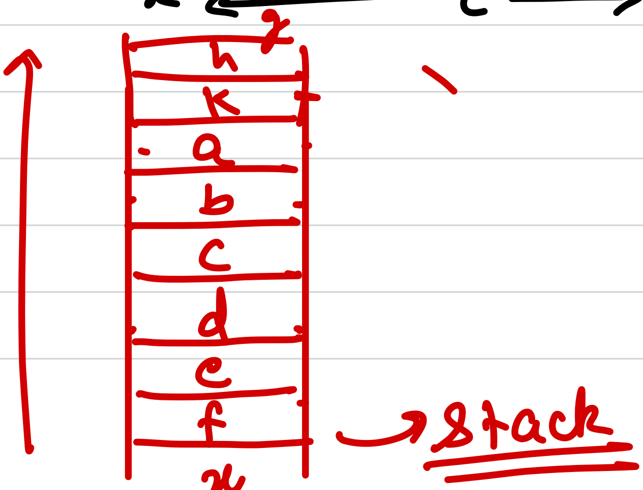
Kosaraju's Algorithm

visited

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

Task → Start Implementing Kosaraju algo





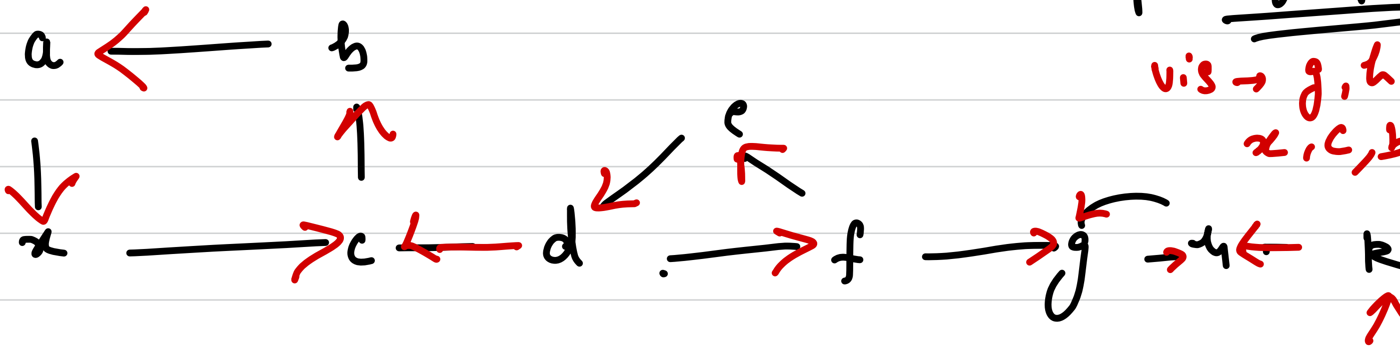
We will start bfs/dfs from an unvisited node.
If we have touched all possible nodes in a go
then we restart with any other unvisited if
it is left.

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

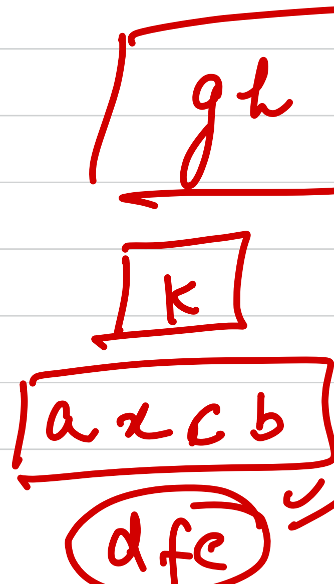
In the phase 2, take transpose of the graph

Start dfs on the transposed graph from the node at the top of Stack.

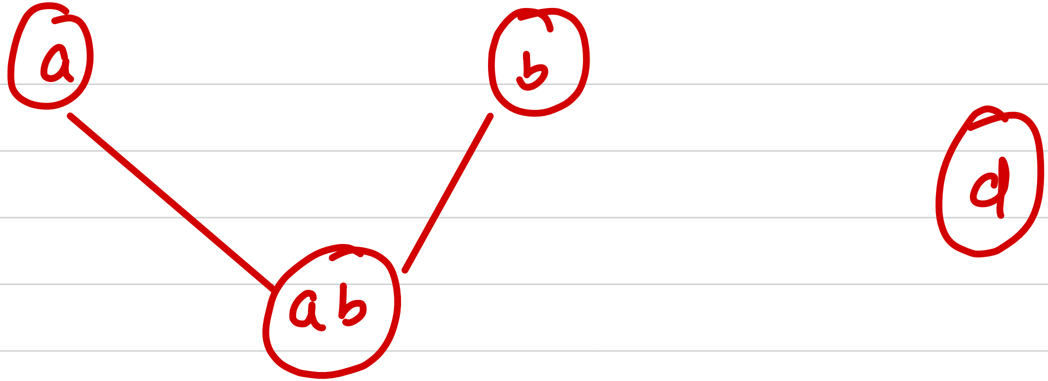
2) transposed graph



vis → g, h, k, a, x, c, b, d, f, e



⇒



Same component

2 DFS / BFS / DSU

10^5

abc

a

ab

b

ab

d 7

a	$\langle 1, 2, 3 \rangle$
b	$\langle 1, 3, 4 \rangle$
c	$\langle 1, \rangle$
d	$\langle 5 \rangle$
⋮	
z	

Q: Given a string of digits of length N. ($N \le 10^5$)
You're standing at 0^{th} index & want to reach $(N-1)^{th}$ index. For each jump you take a cost of 1 unit.
and 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 the min jumps.

0 1 2 1 3 4 4 3 → **4**

0 1 2 1 3 4 4 3

worst case → you need to touch every digit only twice

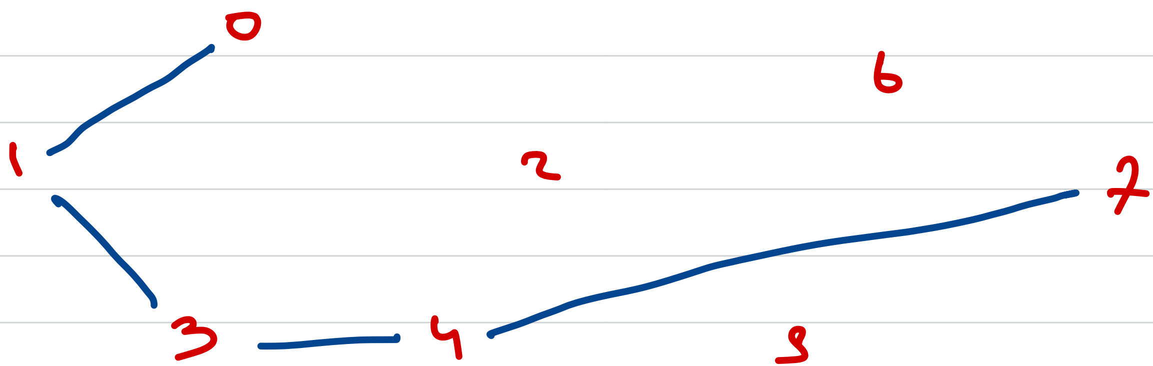
→ 19

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

0 1 2 1 3 4 4 3

0 1 2 3 4 5 6 7

$0 \to 1 \to 2$



$$(i, j)$$

$$\Big\downarrow i$$

$$i+1 \qquad j$$

$$\swarrow \qquad \searrow$$

$$(i+2, j) \qquad\qquad (i+1, j-1)$$

Disjoint Set Union

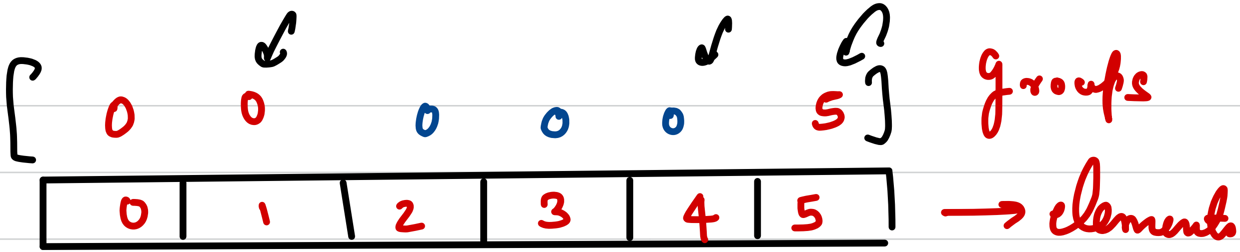
Q. Lets say we want to create clusters

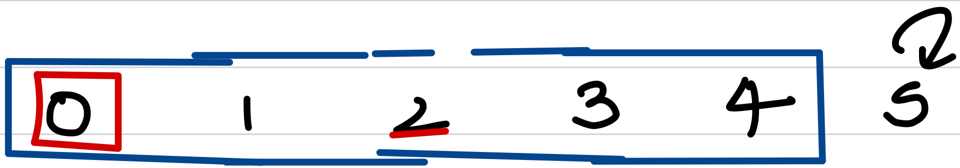
We have a set of elements and we need to group them. Sometimes you might be asked to return the group of any element belongs to.

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

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

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





union(0,1)

union(2,3)

union(3,4)

uinon(1,3)

find(1) → 0

find(5) → 5

find(4) → 0

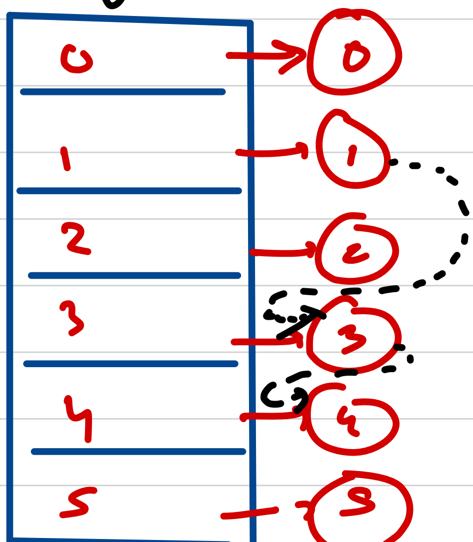
```
int find(x){
    return par[x];
}                                  → O(1)

void union(a,b) {
    a = find(a);                   → O(n)
    b = find(b);
    for i in 1..n {
        if par[i] == b
            par[i] = a
    }
}
```

Can we optimize?

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

map fail



union(1,3)

doubly list

find / get → O(n)

union → O(1)

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

amortized

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

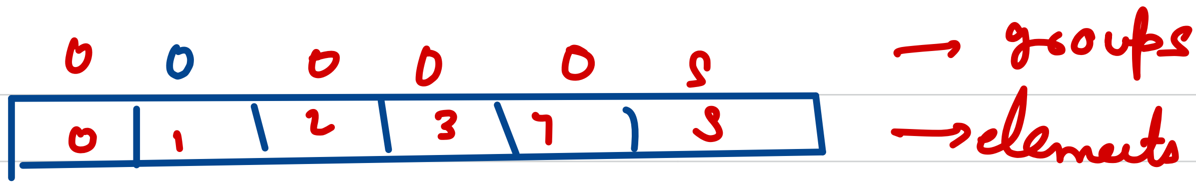
Somehow optimize updating the parent.

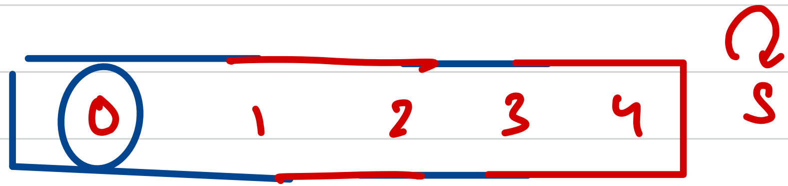
Now let's say we will not union $b \rightarrow a$ directly

We can maintain sizes of the group, and

we will always add smaller group to

the bigger one.





ops → 1

union(0,1)

union(2,3)

union(4,0)

union(2,4)

What is the benefit

1

1

⋮

1

2

2

2

⋮

2

4

4

⋮

4

8

⋮

union by size

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

1

2

4

8

⋮

2^k

$k \rightarrow \log n$

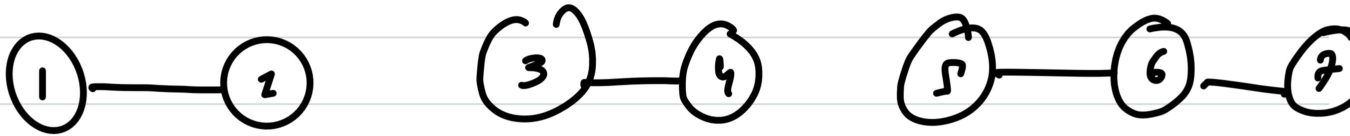
$$1 \times \frac{n}{2} + 2 \times \frac{n}{4} + 4 \times \frac{n}{8} \ldots\ldots$$

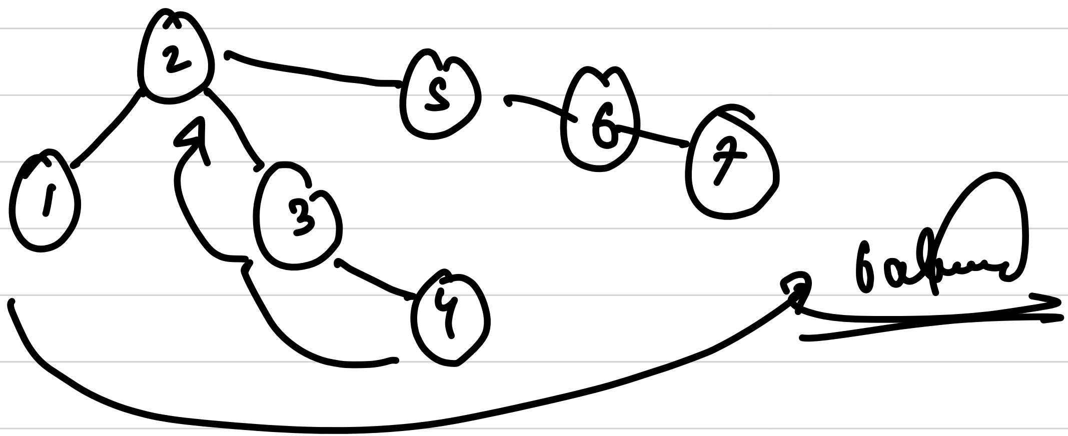
n

→ union → $\log n$

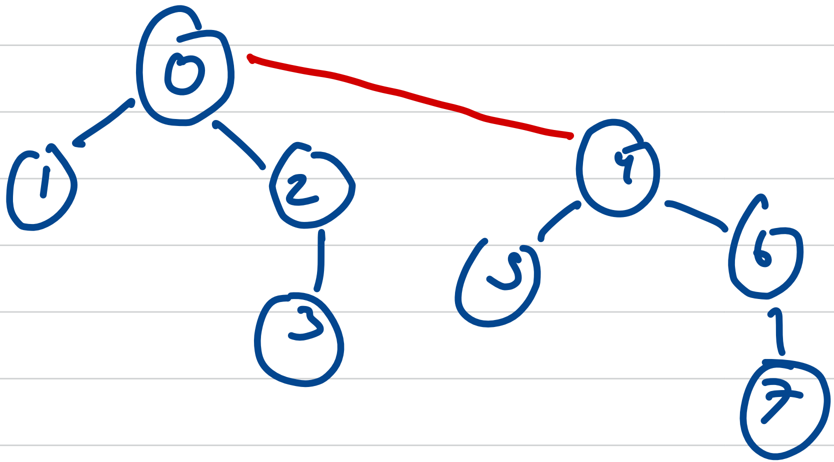
get → $\log n$

Union by size

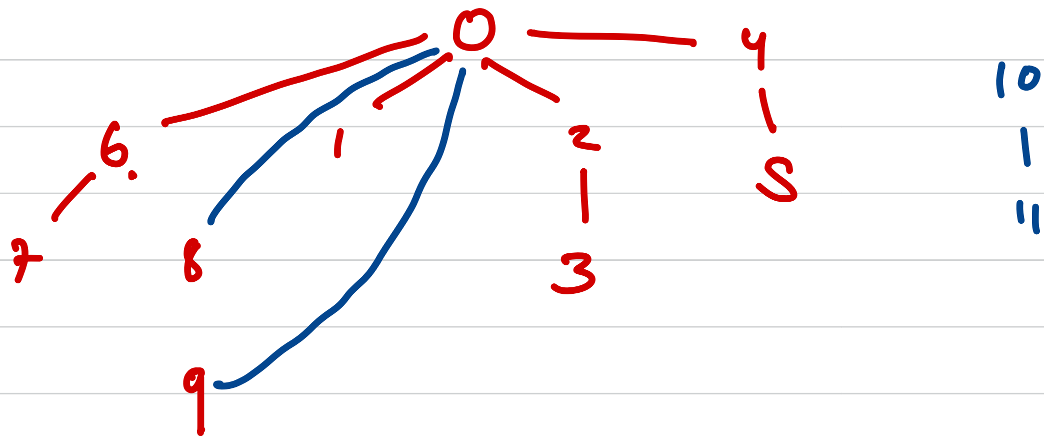




union



0
1
2
3
4
5
6
7



Size

1
2
4
8
16
⋮
2^k

$\log n$

inverse ackl

$\log^* n$

union by rank

path compression

$\log^* n \rightarrow$ extremely slow growing func^n

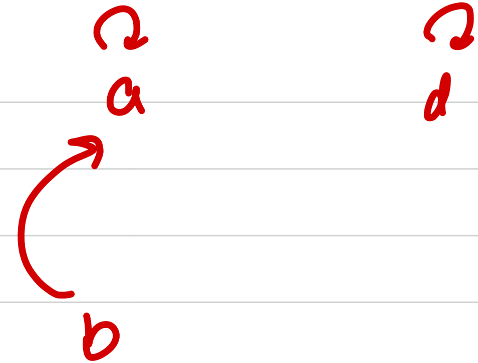
$\log^* n$ no. of steps we need to take $\log_2 n$ on the value of n to make it smaller than 1.

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

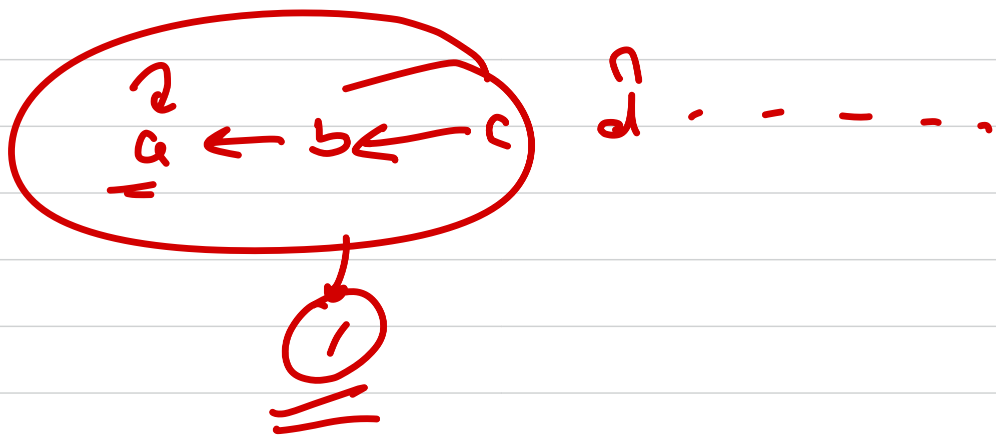
$\rightarrow 1$

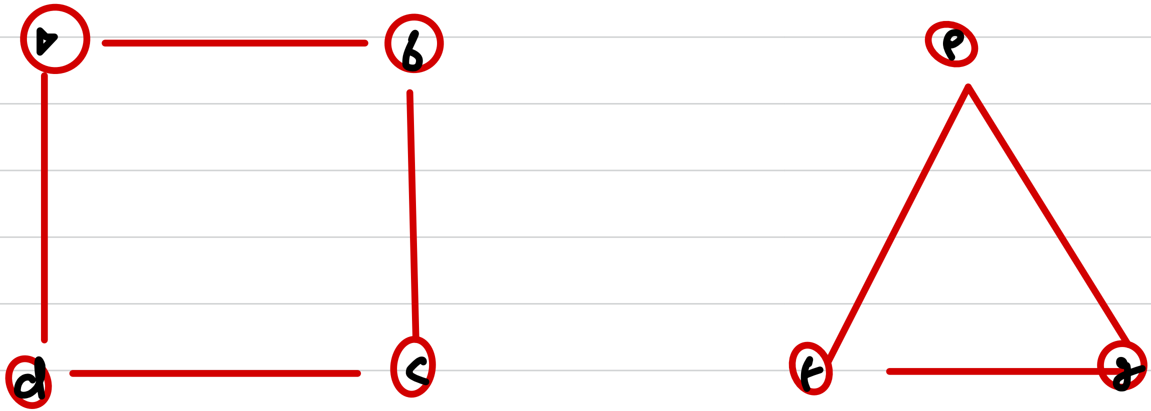
$n \rightarrow 2^{16}$

a
ab
d
b



ab
bc
abc




a
b
c
d
e
f
g


a
b
c
d
e
f
g

Q2 Given, n persons standing in a row at pos [1-n]. We can do 2 ops -

1. $-x \rightarrow$ remove the person at pos x.
2. $?\ x \rightarrow$ find the nearest person to the right of x that is still present (means not removed)

$\boxed{\begin{array}{l} n \le 10^6 \\ q \le 10^6 \end{array}}$

$n = 5$ $\quad q = 3$

-2

-3

$?\ 1 \longrightarrow \underline{\underline{4}}$

$1 \quad \cancel{2} \quad \cancel{3} \quad 4 \quad 5$

? 1

− 2

? 1

union(x, y)

$x \to y$

$-x \to$ union (x, x+1)

? x $\to$ get (x+1)

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

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

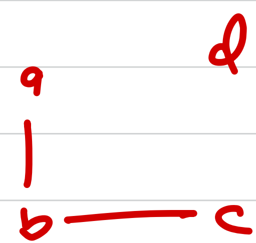
get a b → tell if a & b are in same component.

after all the queries no edge will be left.

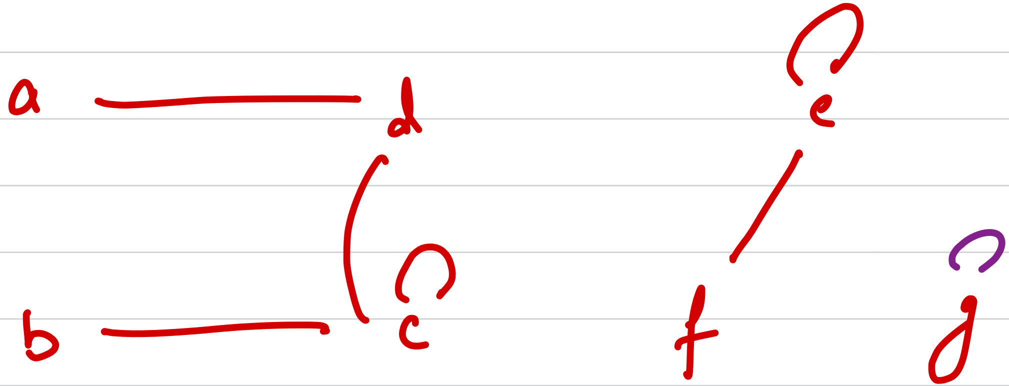
$V \le 5 \times 10^4$

$e \le 10^5$

$q \le 15 \times 10^4$



Offline query



remove a d

union d c

union f e

get b c —— yes

get a f —— NO

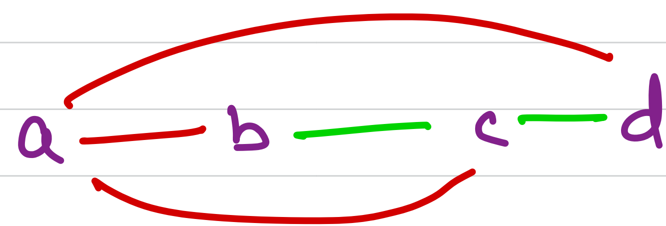
remove c b

get b c —— NO

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

src - a
dest - d

$v \le 10^5$
$e \le 10^5$



Do BFS from both sides

first do bfs from src - dest with only red edges

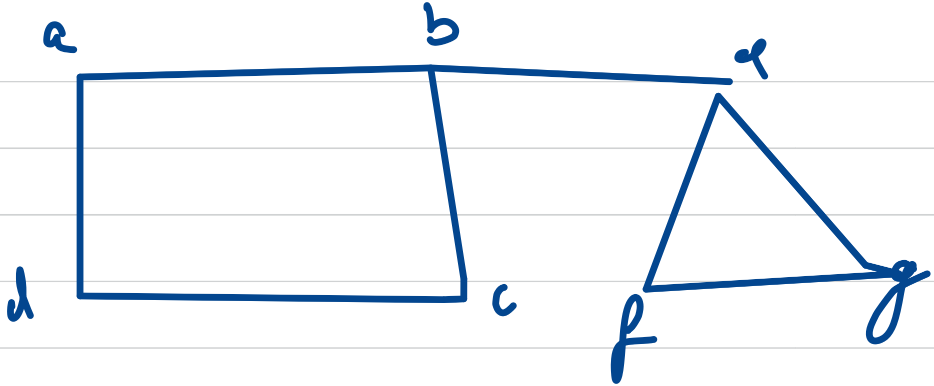
second do bfs from dest - src with only green edges src dst

a s d

result

sd

Ex



G1 →

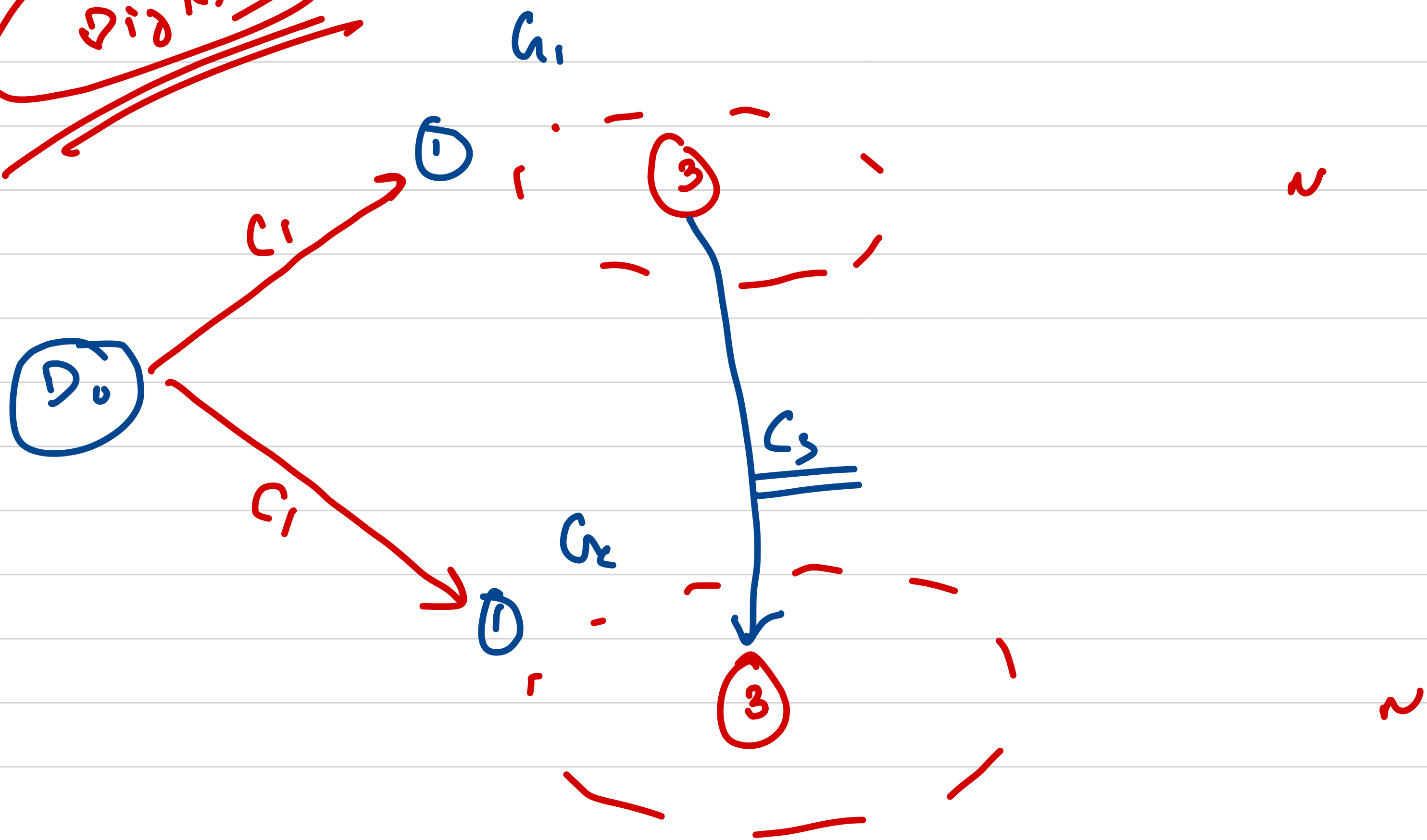
there is an edge from $v_1 \to v_2$ if $h_{v_1} \leq h_{v_2}$

start → cost

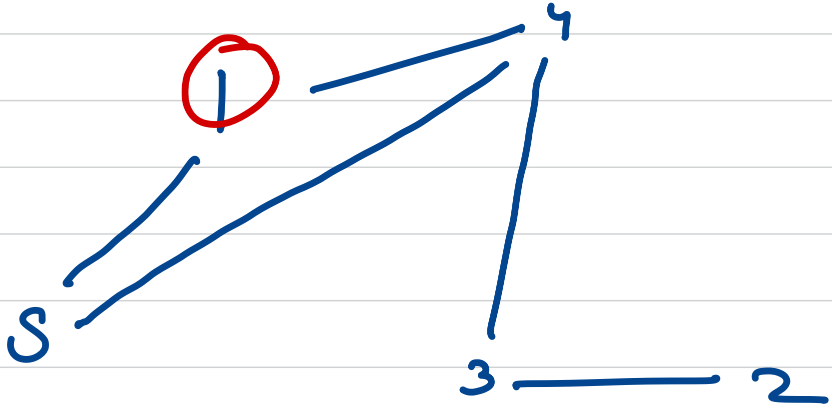
G2

$h_{v_1} \geq h_{v_2}$ → down the hill

Dijkstra



BFS



We have to take care of smaller element first

1, 4, 3,

priority-queue

queue<int> qu;

priority-queue<int> pq;

[wrt, $\overset{2}{\text{wrf}}$, $\overset{2}{\text{er}}$, $\overset{3}{\text{ett}}$, $\overset{1}{\text{rftt}}$]

wrt 2
wrf 2

wrt
er

wrt
ett

er 2
ett

t f

e w r

t → f
e → r
w → e
w → r
r → t

w e r t f

[wrt, wrf, er, ett, rftt]

wrt
wrf

t → f

wrt
rftt

w → r

wrt
er

w → e

O(n²)

er
ett

ert
rftt

n ≤ 100

w r

t f e

wertf

Topo

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

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

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

3	1	4

1

2

Yes

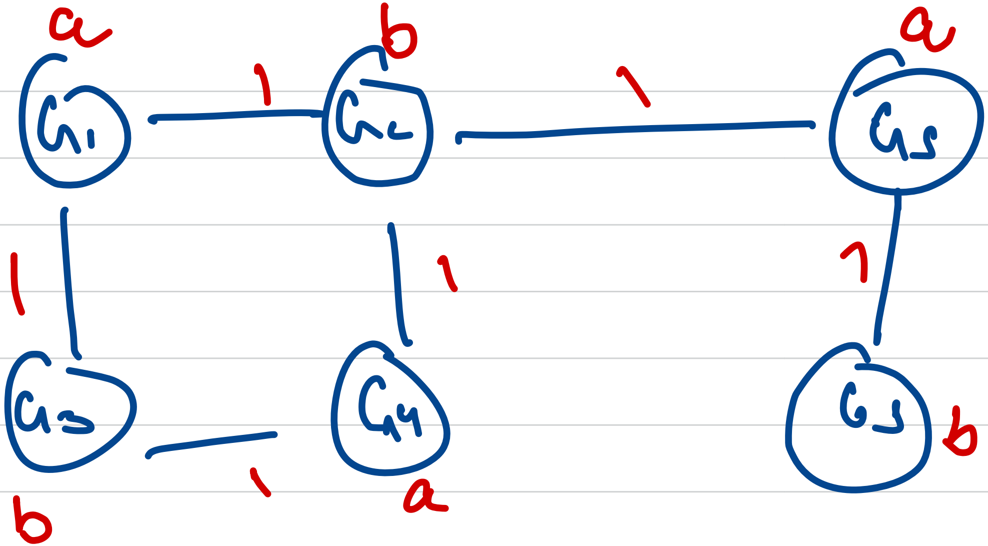
if $M_{ij} == 0$ then $B_i == B_j$

$M_{ik} == 1$

$B_i \; != B_k$

lazy entity

$M_{ij} == 1$



2 coloring

Qn Given a tree, select 2 nodes, we need to satisfy two conditions

(1) Subtrees rooted at the 2 selected nodes must be disjoint.

(2) Sum of node values → max

f(node, c .)

max value you

can get by considering

subtrees of node &

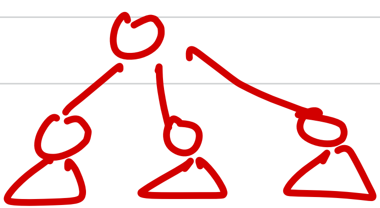
taking c no of nodes

$f(n, 1)$ $f(n, 2)$

sum of nodes

$$f(n,1) = \max \begin{cases} \text{sum of node}(n) \\ f(c_i, 1) \end{cases}$$

$$f(n,2) = \underline{\underline{\max}} \begin{cases} f(c_i, 2) \\ \text{or} \\ \text{calc } f(c_i, 1) \rightarrow \max,\ 2^{nd} \max \end{cases}$$

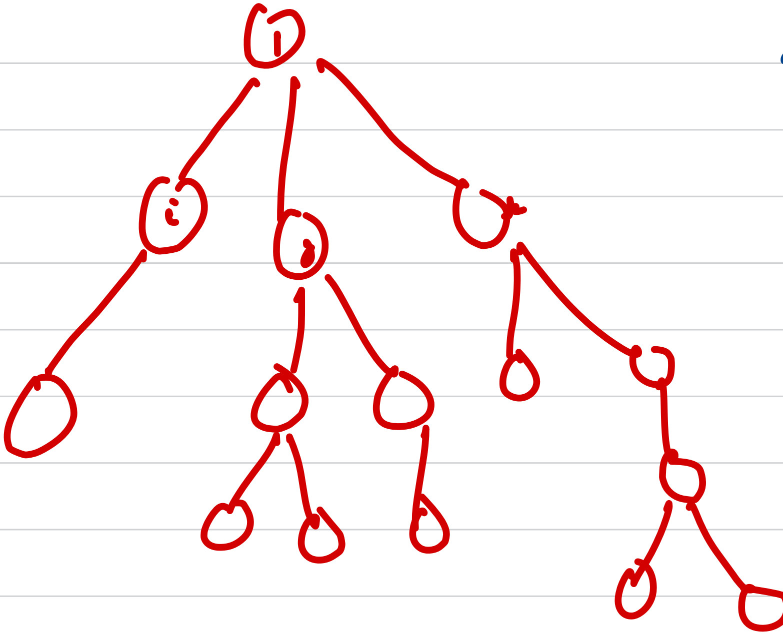


Base

u → ley nu

$$f(u,1) = a_u$$

$$f(u,2) = -\infty$$

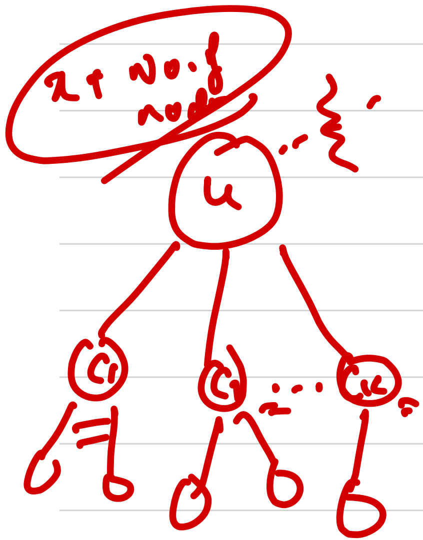


$f(1)$: sum of distance $(1, i)$ $\forall i$

$g(i) =$

$\underline{f(n)}$

$$g(u) = \sum_{i=1}^{k} g(c_i) + \text{nodes}(c_i)$$



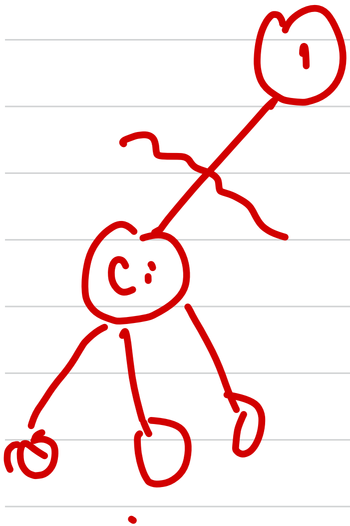
$$f(1) = g(1)$$

nodes(n) → count of nodes of subtree rooted at n

Rerooting

$g(c_i)$

$f(c_i)$??



$p(1, c_i)$

$$p(1, c_i) = f(1) - (g(c_i) + node(c_i))$$

$$f(c_i) = g(c_i) + p(1, c_i) + N - node(c_i)$$