GRAPH — undirected

- directed

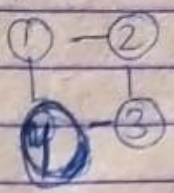
weighted graph

Representation : (undirected graph)

n → nodes, m → edges

m lines

1	2
2	3
1	4
3	4

→ 

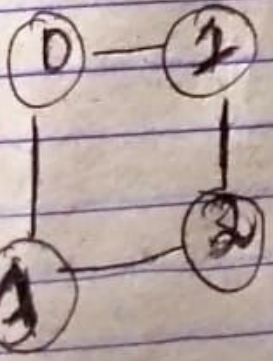
can create Adj. matrix OR Adj. List

space → $O(n^2)$ & $O(n+2m)$

→ convert input format to adjacency list

Input FORMAT

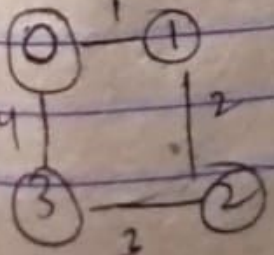
n, m, m×2 matrix



n = 4, m = 4

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

How to store weighted graph?



n = 4, m = 4

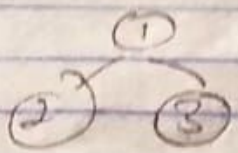
0, 1, 1
1, 2, 2
2, 3, 3
3, 0, 4

② Traversal –

PAGE NO.
DATE: 11.

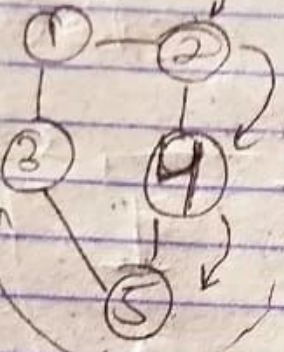
DFS BFS

→ keep unvisited Track Before



④ { Graph with 2 components

DFS Traversal : (Recursion Round)

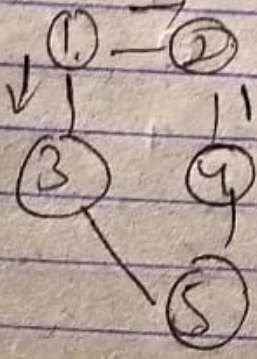


1, 2, 4, 5, 3

(stack) of Recursion

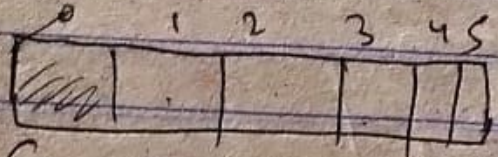
Time → O(N) + O(2M) ≃ O(N)
↳ nodes → edges

BFS Traversal (Breath cover)



1, 2, 3, 4, 5

(queue)



vis

boolean

Time →

O(N) + O(2M)

FOR multiple components

(3) BFS Problem Solving

Start = 3 ——— End = 30

arr = {2, 5, 10}

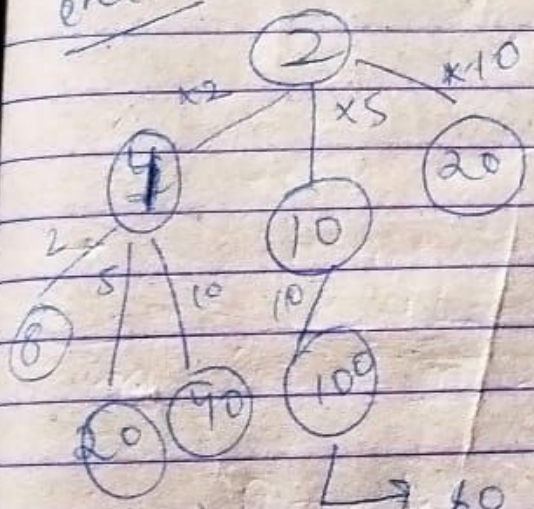
Min operation to Reach from start to end

↓

means multiply

start = 2

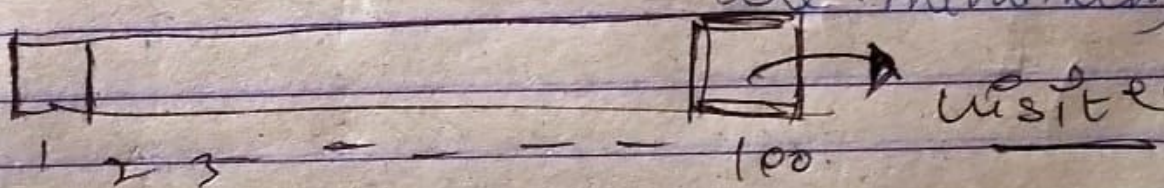
end = 100



eg: 3 × 2 × 5 → 30 (2 operation)

3 × 10 → 30 (1 operation)

↳ so stop (2 operations are minimum)



Time Com → O(End - Start)

↳ maximum visiting

(4) BFS Problem (Rotten oranges)

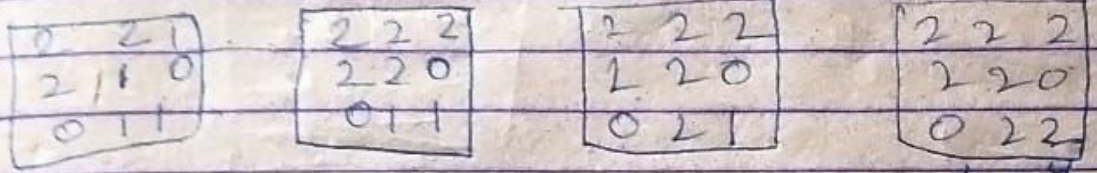
0 → empty orange, 1 → fresh orange

2 → rotten orange

(min time in which all oranges will be Rottenly)

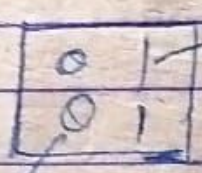
grid = Input → $\begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$ output → 4

TC (m x n)



t=1 t=2 t=3 t=4

(5) The maze : ball can Roll through empty spaces by {up, down, left, right}



→ wall

↓ empty space

(not stop Rolling until hitting a wall).

(Lintcode 787)

Inputs start (r, y) end (r, y)

maze [][] (2D)

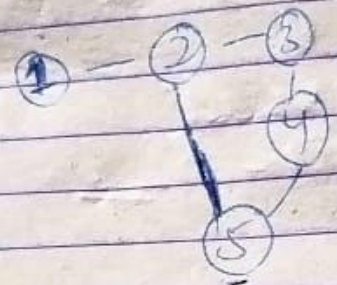
=

(5) Cycle Detection

(Directed, undirected graph)

BFS/DFS

(I) DFS (undirected graph) cycle detection



DFS(1,0) → node, parent

DFS(2,1)

DFS(3,2)

DFS(4,3)

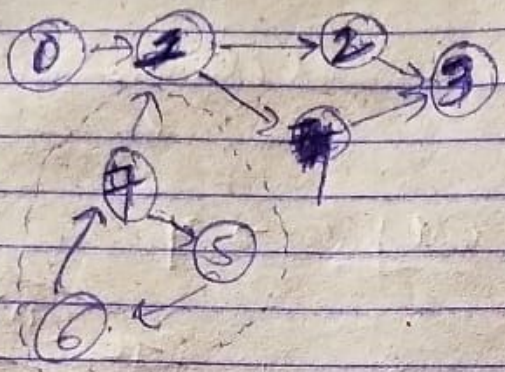
DFS(5,4)

2,4

DFS(5,2) { now 2 is visited & 5 is not parent of 2 (return true)

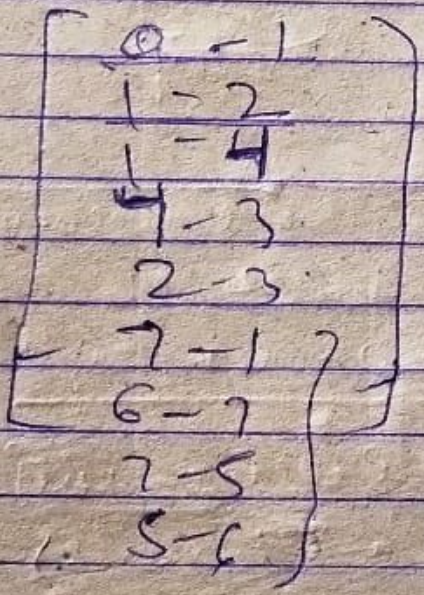
2 is visited & 5 is not parent of 2.

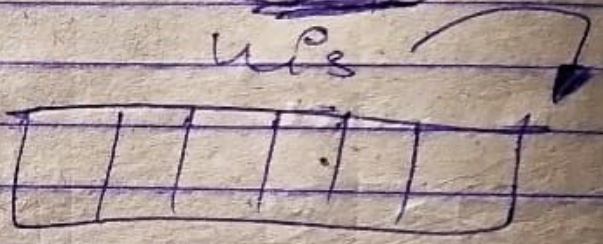
(6) Detect cycle in directed graph :



cycle = True

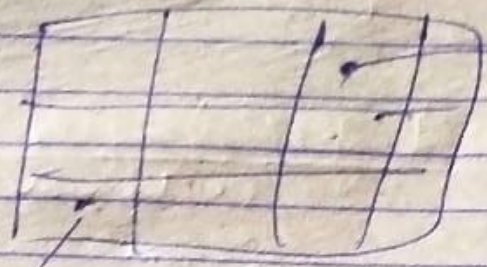
different type component





marked nis

⑦ Min steps by knight

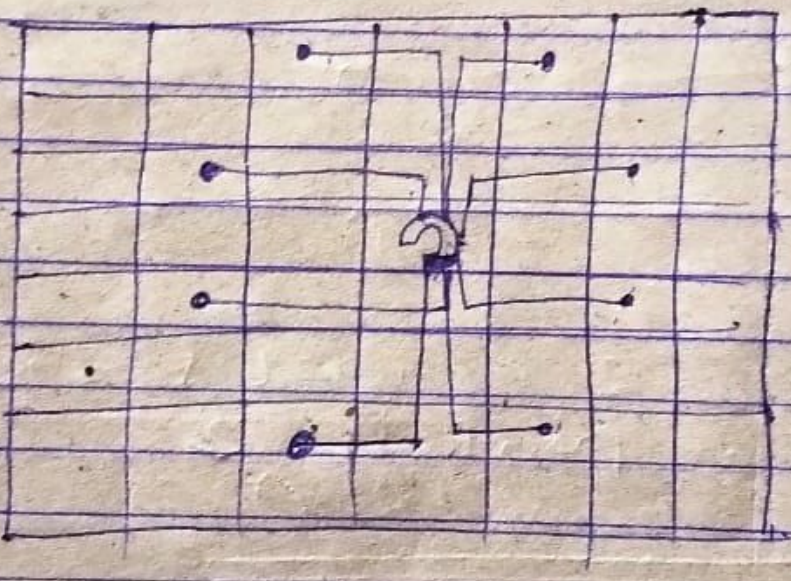


Start

knight (2 1/2) moves can Run

end

N*N board



dx

dy

8 Positions

8 possible Position we can explore

TC

$O(N^2)$

Space

$O(N^2)$ → visited

: {poon, plee, same, poie, plea, plie, poin}

↓ visited

(toon, 1)

(poon, 2)

(poin, 3)

(poie, 4)

(plie, 5)

(plee, 6)

(plea, 7) → when get target string return step

(BFS) word Ladder 1

start = toon
end = plea

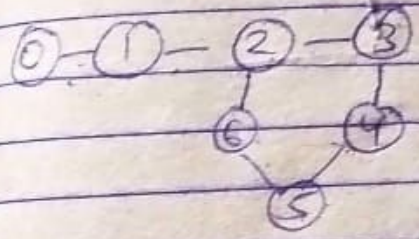
plie & plee → True

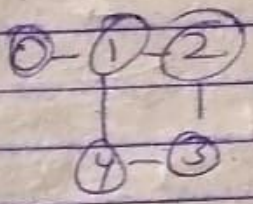
plie & plea → false

(9) Bipartite graph : graph whose vertices can be divided into 2 disjoint set U, V.

: when we can color all the nodes using 2 colors such that no 2 adjacent have same color is Bipartite.

: Acyclic or even length is Bipartite
: odd length is not Bipartite





Note :
If not a bipartite then odd length
sure

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

0-1
1-4
1-2
2-3
3-4

DFS(0) color 0

DFS(1) color 1

If at any time adj. nodes have same color return false.

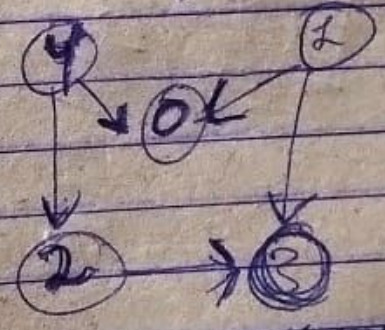
① TopoSort: Linear order of vertex so u appears before v where edge: $u \to v$

TopoSORT can be applied on →
1) directed graph
2) no cycle should be there.
so possible for (DAG).

TopoSORT : (method)
↳ Stack method (DFS) ✓
↳ Kahn's (BFS) =

STACK METHOD	KAHN'S
D(1) / D(2) / D(3)	(Indegree concept)

1 → 2 → 3



1, 2, 3

TopoSORT concept.

14023 → TopoSort

(1) Problem Solving (topo)

(207)

Q = Course Schedule

given a number of courses & PREREQUISITE matrix where (a.i) & (b.i) represent course where b can only be done after a.

topoSort(int numC, int[][] prereq){

if you can create a topo sort array of numC length then course schedule can be created.

}

Q- Alien Dictionary Problem

Topo sort

0/1 BFS

the graph having only 0/1

{ then 0/1 BFS concept can be applied }

(12) Dequeue DS is used in 0/1 BFS

Agenda - given a graph where every edge has weight as 0 / 1

Find the shortest path from source vertex to every other vertex.

{Image of 0/1 BFS for this} →

Implement → (13)

Intuition maintain the sorted order in Deque DS.

⇒ what if we have more type of weight 1, 4, 9, 8, 5 - - - - - - taken

→ Priority queue is used (Dijkstra)

Intuition

Because all previous one we will have to consider the element from queue which have shorter distance to move forward.

{Greedy kind of Approach} $O(N \log N + E)$

↳ (single source shortest path) | two nodes only

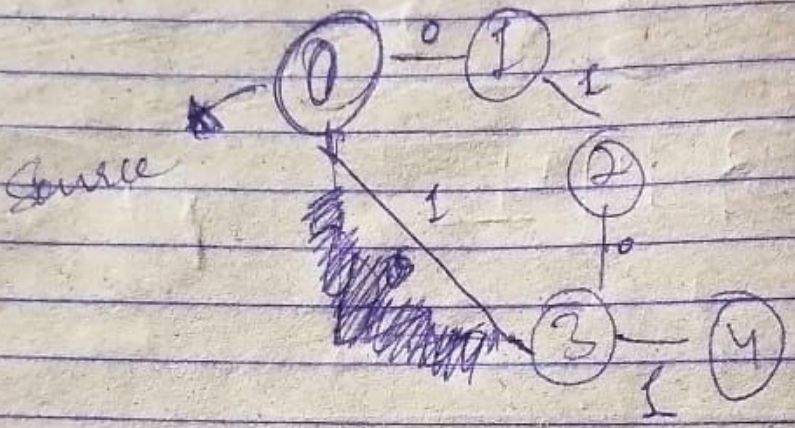
PAGE NO.
DATE : /

dist

0	1	2	3	4
0	1	1	2	

(13)

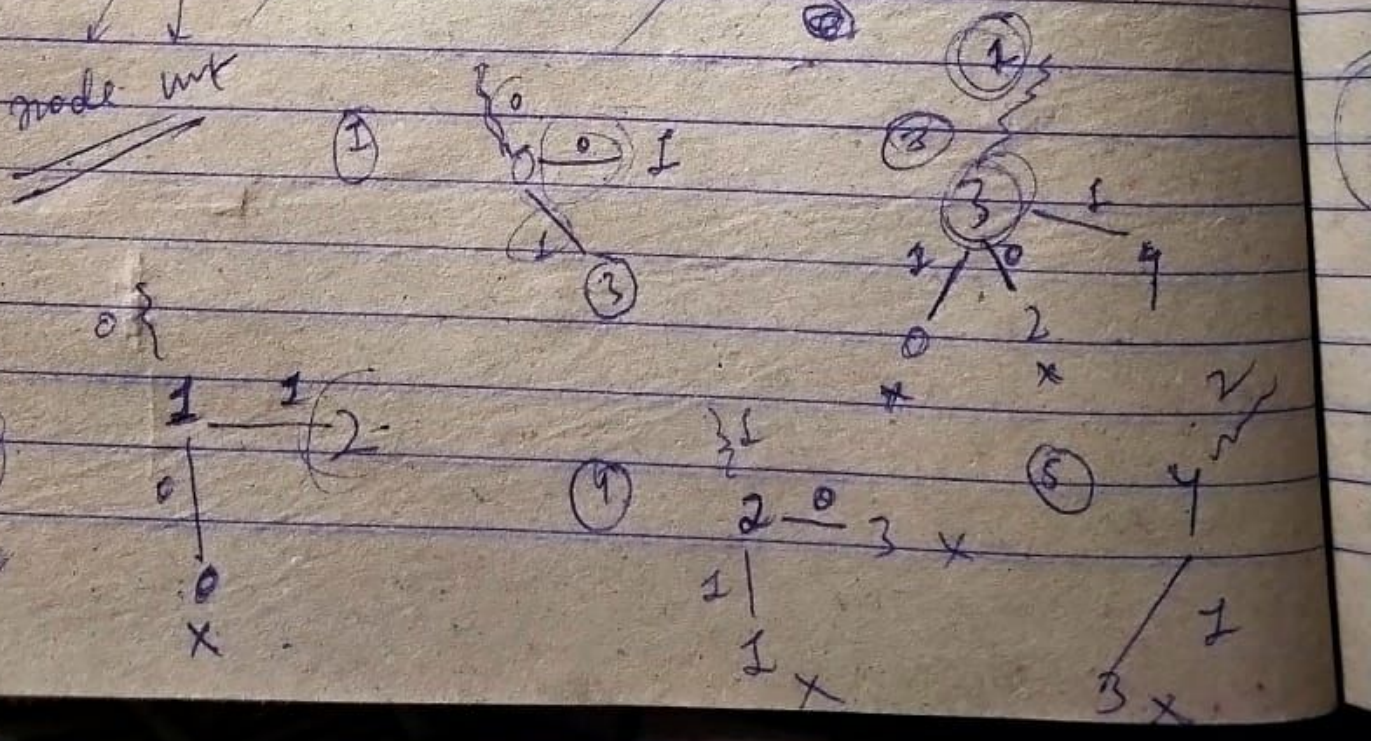
- 0, 1, 0
- 1, 2, 1
- 0, 3, 1
- 3, 4, 1
- 2, 3, 0



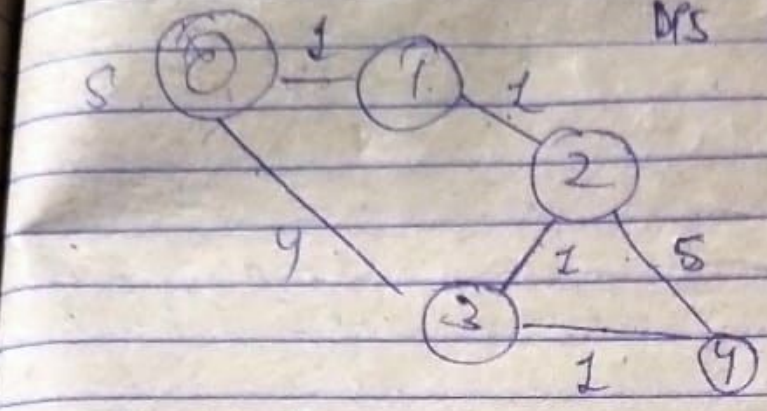
Dequeue (insert/delete from both sides)

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

node wt



(14) Single source Shortest Path
Dijkstra Implementation -



	0	1	2	3	4
DIS	0	∞	∞	∞	∞
		1	2	4	7
				3	4
	0	1	2	3	4

ans

- 0, 1, 1
- 0, 3, 4
- 1, 2, 1
- 2, 3, 1
- 2, 4, 5
- 3, 4, 1

(5) (4,4)
X

(6) (3,4)
X

(7) (4,7)
X

(4,4)
(3,3)
(4,7)
(2,2)
(1,1)
(3,4)
(0,0)

dis

node

Priority Que

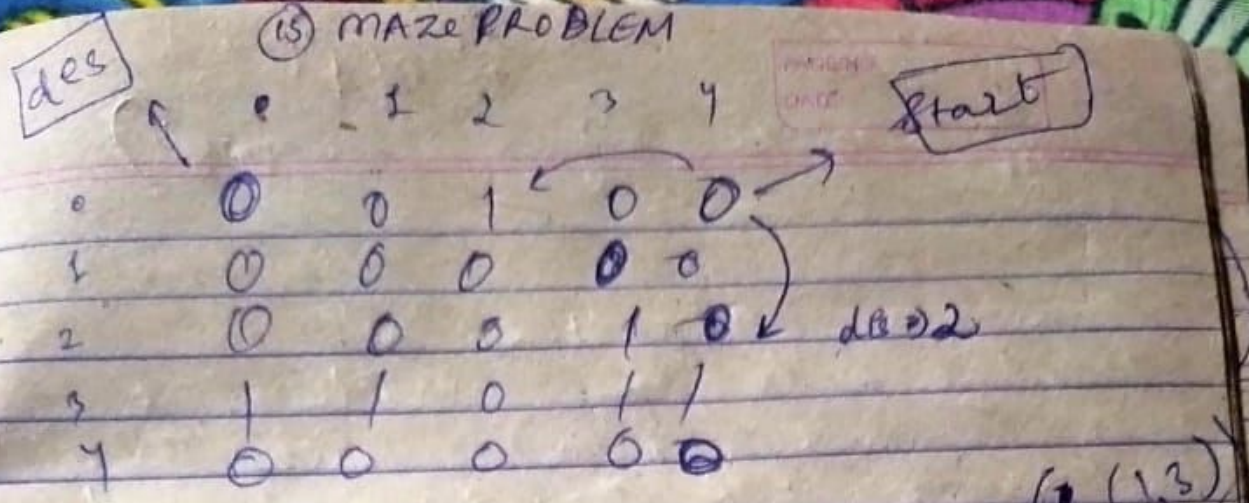
(1) (0,0)

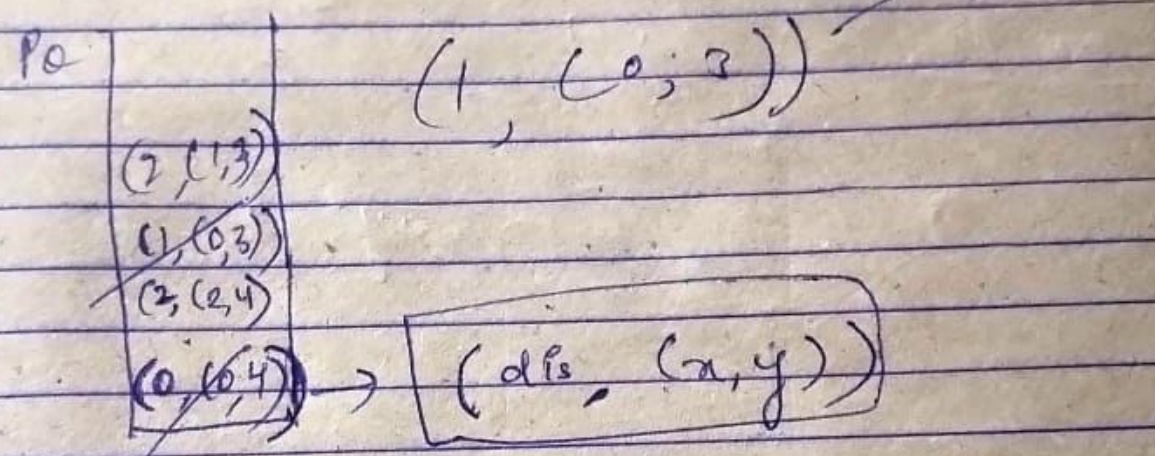
(2) (1,1)

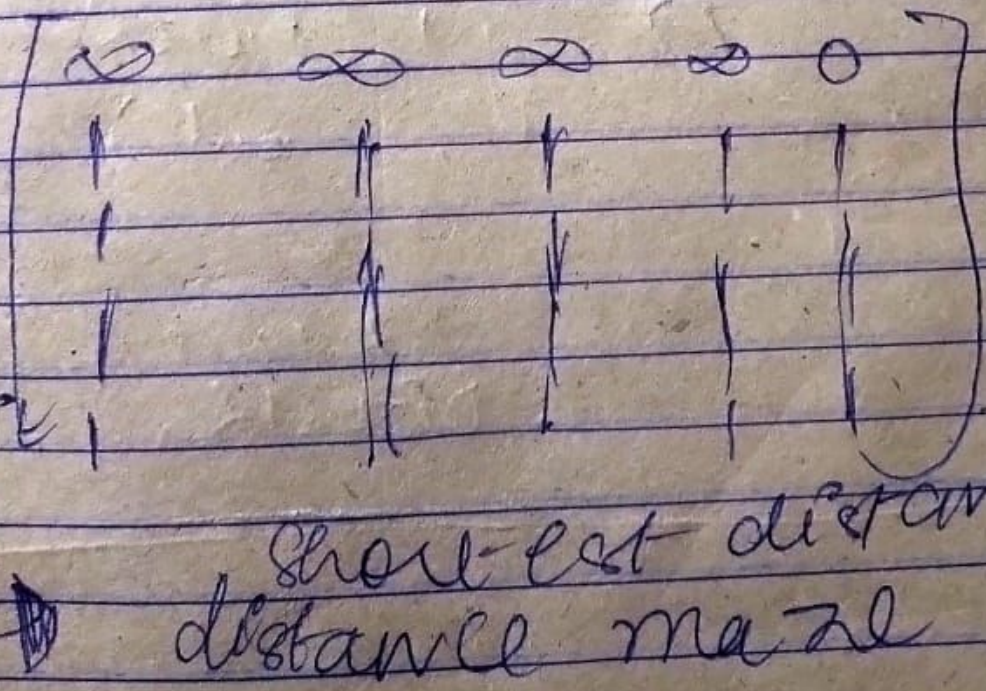
(3) (2,2)

(4) (3,3)

(15) MAZE PROBLEM







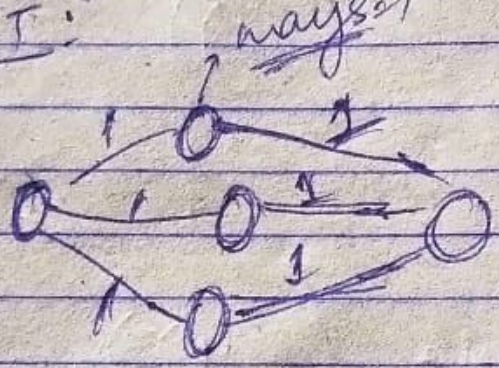
Shortest distance
distance maze

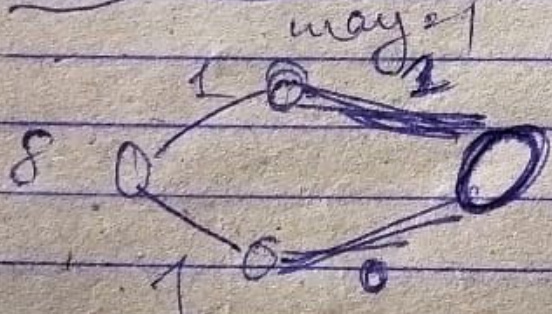
(16) Question

Leetcode (No. of ways to Arrive at Dest)

As same as Dijkstra Just catch the case in which the same distance (path) is taken Track the ways for Reaching at node.

CONCEPT:





(Filtered out)

↓

as we filtered in Dijkstra

(17) Disjoint Set Union

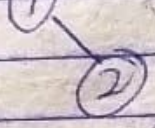
Use Case - To fig. out if 2 nodes belong to same component

operations are →

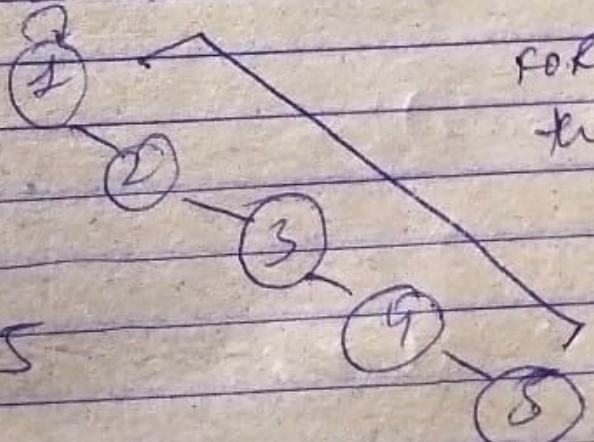
1) union 2) find Parent

Terms:

ultimate Parent of 2 is (1)



Now in some case of union we end up creating a skew Tree like This:

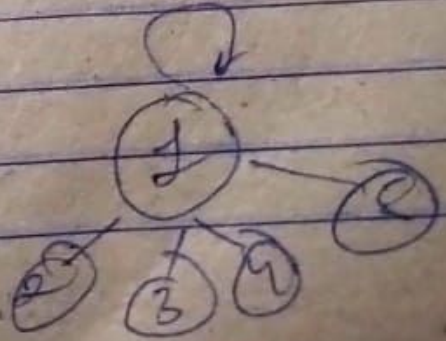


for checking the ultimate Parent it will Take O(n)

n=5

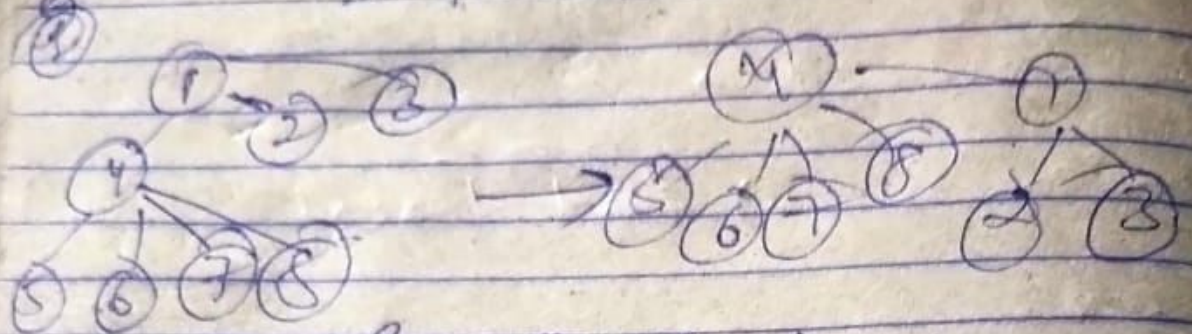
Optimization Techn-

PATH COMPRESSION :



so connect Pow

(18) So At each step Path compression will change Graph Structure →



{union by PATH Compression + size compression}

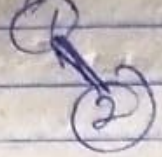
```
class DSU {
    ArrayList<Integer> parent, size, rank;
    DSU(int n) {
    }
    int findPar(int node) {
    }
    void unionSize(int u, int v) {
    }
    void unionRank(int u, int v) {
    }
}
```

(19) Union by Rank

(a) union(1, 2) → Rank

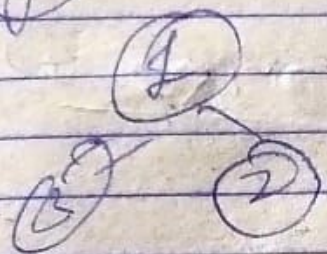
1	2	3
1	0	0

1 → 0
2 → 0
so now
1 → 1

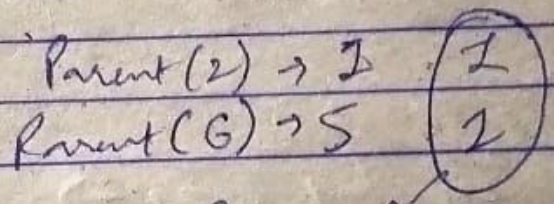


1	2	3	5
1	0	0	1

(a2)



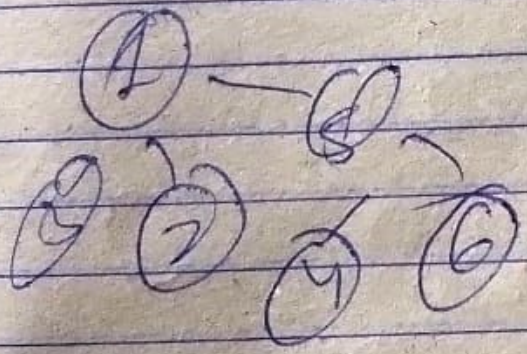
union (2, 6) → Parent(2) → 1
Parent(6) → 5



Rank is equal

so

1	5
2	1



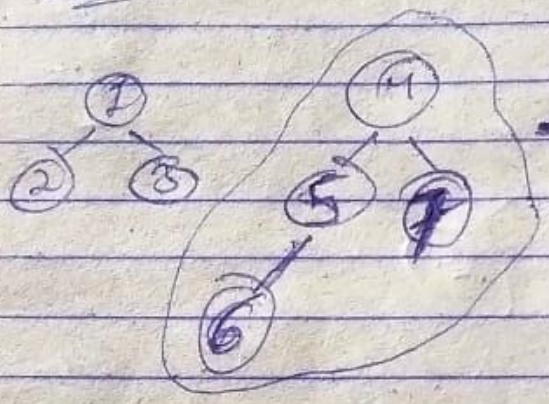
(20) Disjoint Union Set

PAGE NO.
DATE:

n=5 (init) → | 0 | 1 | 2 | 3 | 4 | → PARENT

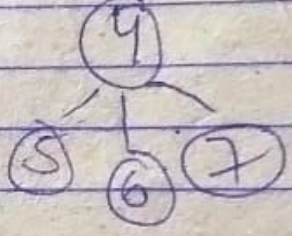
0 1 2 3 4 } each node Parent's are same

Find Parent

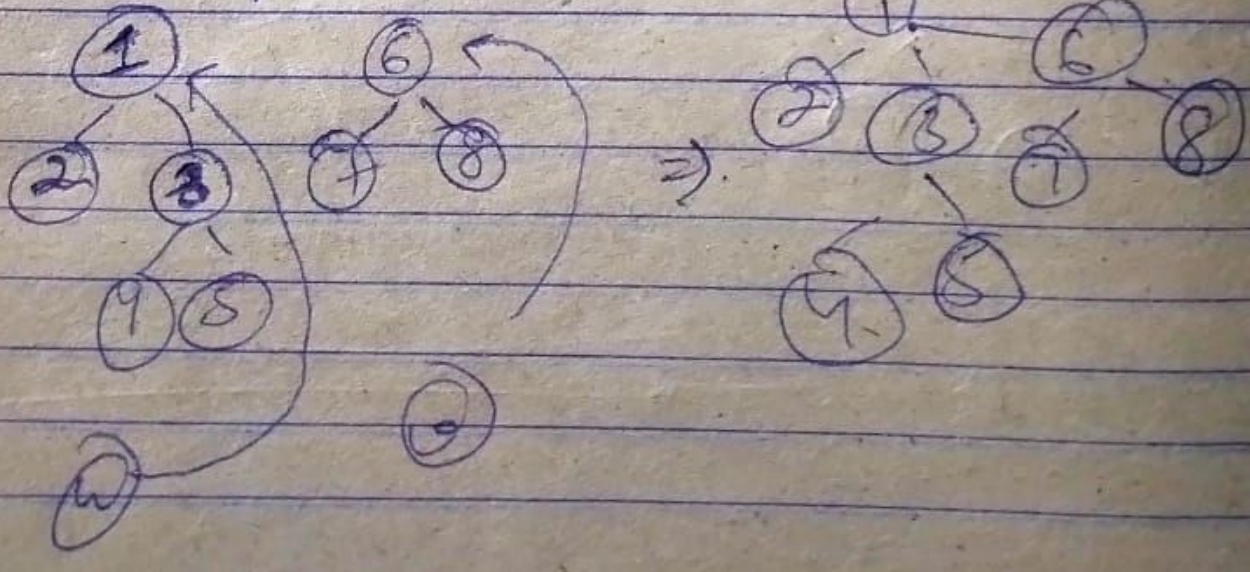


FindParent(6) → ans 4

PATH COMPRESSION



union By Size →



(I) PATH compression only.

O(log n)

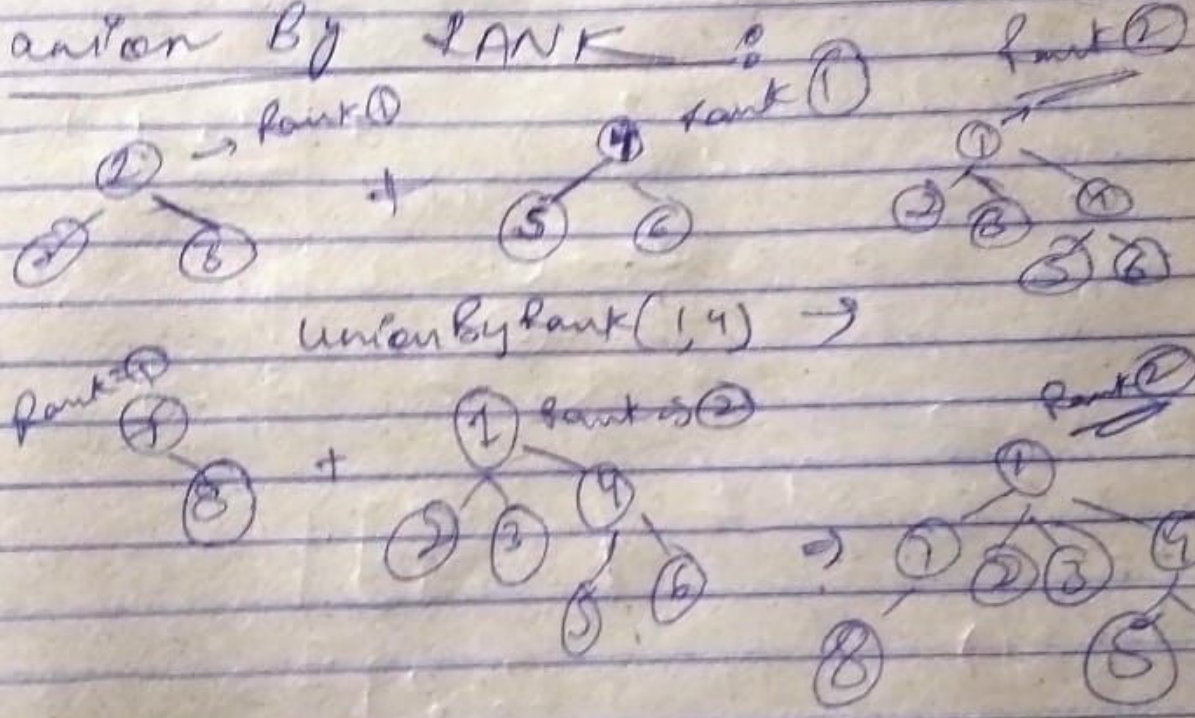
(II) PATH COMPRESSION + SIZE

O(4 * ALPHA)

(III) PATH COMPRESSION + RANK

O(4 * ALPHA)

union By RANK :



→ No. Of Islands 3 (DSU) approach

↳ Shortcode

(22) Example

n=3, m=3

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

Points

$$\begin{bmatrix} (0,0) \\ (0,1) \\ (2,2) \\ (2,1) \end{bmatrix}$$

DSU → (9)

(1) (0,0)

make mat[0][0] = 1

Island = 1

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

(2) (0,1)

← found (1)

>0

(0,1) is (1) → (1) — union — (0) — (1)

(0,0) is (0) → (0)

Island = 1

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

(3) (2,2)

Island = 2

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

(4) (2,1)

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

(2,1) (7) → 7

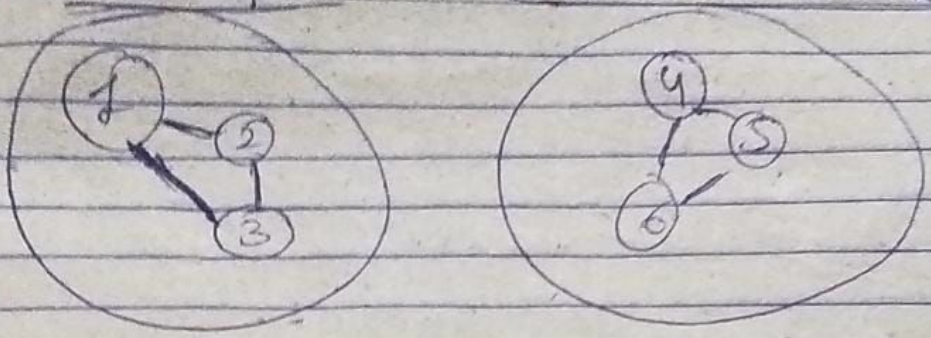
(2,2) (8) → 8 — union — (7) — (8)

Island = 2

(23) 25D codeforces (DSU)

Roads not only for Berland

Quest Explain — (A component will have n-1 output)



Now connect all the components by Removing any one edge & use only this existing edge

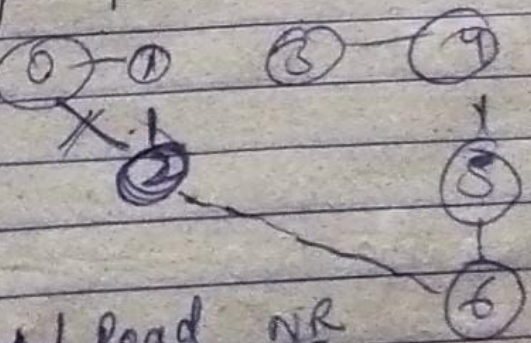
→ Store extra Roads while connection

→ Find no. of component with col+$ Parent.

→ use adjacent unique parent to build Road.

eg

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



old Road NR

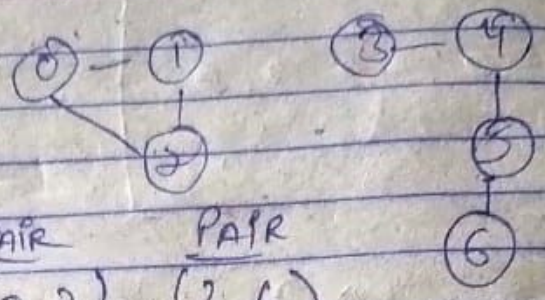
0-2 2-6

so both component connected

(7) → edges

(24)

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



PAIR PAIR

ans (0,2), (2,6)

DSU

(1) union(0,1)

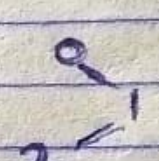
PAR(0) → 0

PAR(1) → 1

(2) union(1,2)

PAR(1) → 0

PAR(2) → 2



(3) union(2,0)

PAR(2) → 0

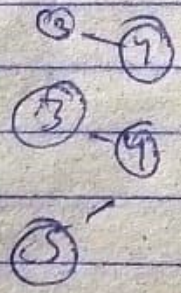
PAR(0) → 0

EXTRA ROAD

(2, 0)

(4) U(3,4) →

(5) U(4,5) →



(6) U(5,6)

PAR(5) → 3

PAR(6) → 6

set → ultimate Parent[0]

{ 0, 0, 0, 3, 3, 3, 3 } = {0}

→ 0 —— edge -1 find par & put in set.

set = {0, 3} components are → (2)

So new connections are (1)

x = 2, so, output (x-1) connections

Now in (x-1) lines output.

dt road ← extRaRoad[i].x + extRaRoad[i].y + (set[i] + set[i-1])

new Rodd

MST (Minimum Spanning Tree)

Graph convert to Spanning Tree by removing edges by preventing from cycle.

Spanning Tree → n-1 edges
→ n nodes

Minimum spanning tree:

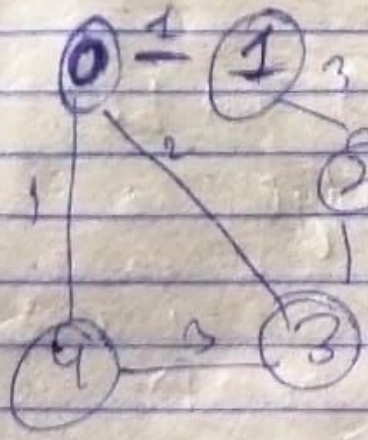
In which cost is minimal

Possible spanning trees

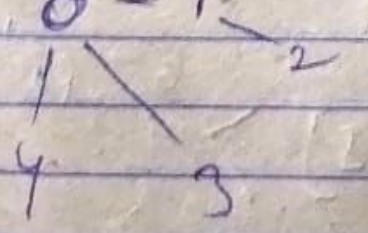
Kruskal's Algo →

Simple sort & add the edges according to DSU concept:

→ Sort acc the weights acc to cost.
→ (u, v) PART of MST iff par(u) != par(v)
→ so union is possibles.



0,1,1 ✓
0,3,2 ✓
1,3,3 ✗
1,2,3 ✓
2,3,4



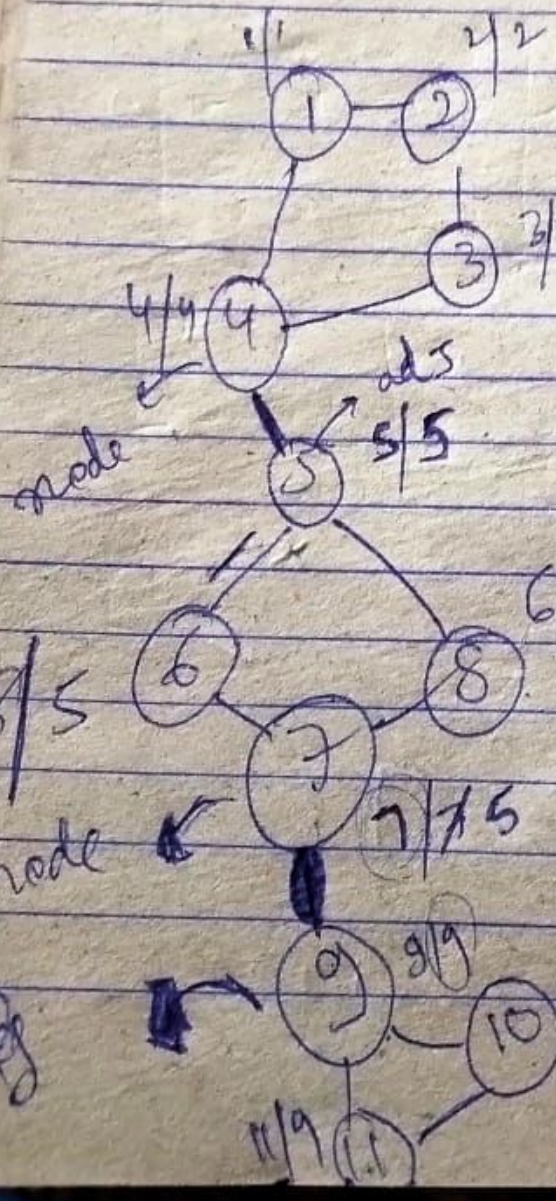
cost = 1+1+2+3

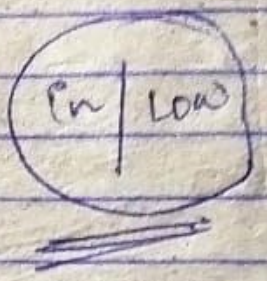
Bridges in a graph :-

It is an edge on whose removal graph gets disconnected.

Terms :

1) time of insertion : time when you reach
2) lowest time : the minimum time around all its adjacent nodes apart from parent.





∴ 6–7 can't be a bridge as at 6 you reach at time 5 & at 7 also via 5→8→7

Formula

low[adj] > In[node] } Bridge

28

Implement simply with DFS

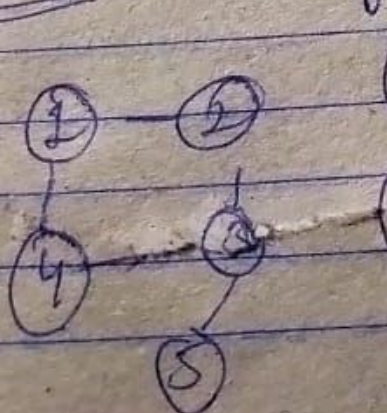
T.C → O(N+E)

IMP NOTE FOR BRIDGE DETECTION

```
DFS ( ) {
    if (not visited) {
        DFS( )
        low[node] = min(low[node], low[adj]);
        if( low[adj] > in[node])
            node + " " + adj
    } else {
        low[node] = min(low[node], in[adj])
    }
```

not reachable by any other Path

Q- Is there an edge b/w c & d so that disconnect (c & d) and then check, if you can visit both the nodes Nodes : if you start from c.



so in DFS ignore when

node == c && adj == d
or
node == d && adj == c

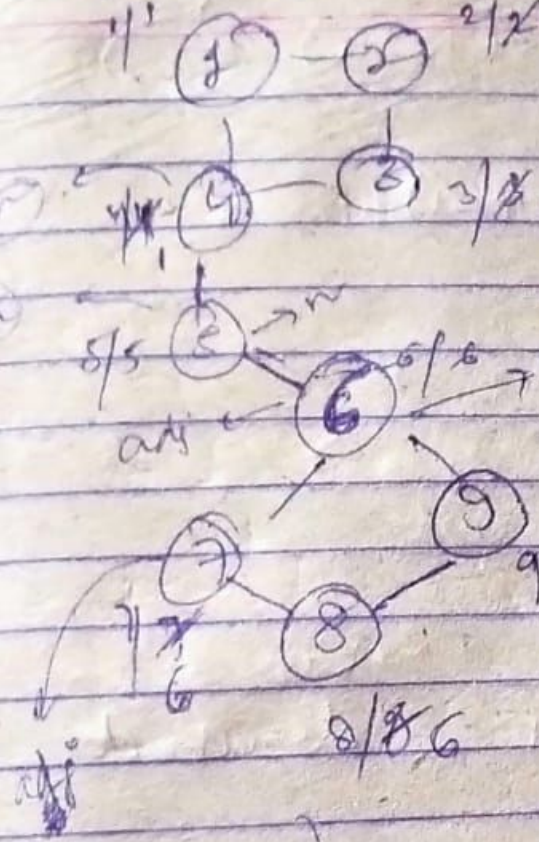
(29)

```
void dfs(int node, ArrayList<ArrayList<Integer>> adj, boolean[] vis, int c, int d) {
    vis[node] = true;
    Iterator<Integer> itr = adj.get(node).listIterator();
    while(itr.hasNext())
    {
    int neigh = itr.next();
    if((neigh == c && node == d) || (neigh == d && node == c)){
      continue;
    }
    if(!vis[neigh])
      dfs(neigh, adj, vis, c, d);
    }
}

boolean isBridge(ArrayList<Integer> adj[], int c, int d){
    boolean[] vis = new boolean[adj.size()];
    dfs(c, adj, c, d);
    if(!vis[d]) return true;

    return false.
```

(30)

(30) Articulation Point

TC (V+E)
SC O(V)

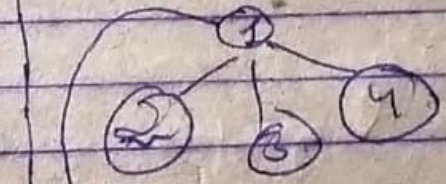


disc[]
low[]

Formula:

low[adj] >= disc[node] && par != -1

edge case



If it is a Articulation Point so check if (child > 1 && parent == -1)

low[adj] = 6
disc[adj] = 6

low[adj] = 6
disc[n] = 5

4
5
6

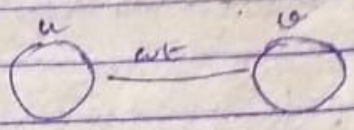
low[adj] = 5
disc[node] = 4

(3) Bellman ford (DP)

(single source shortest path)

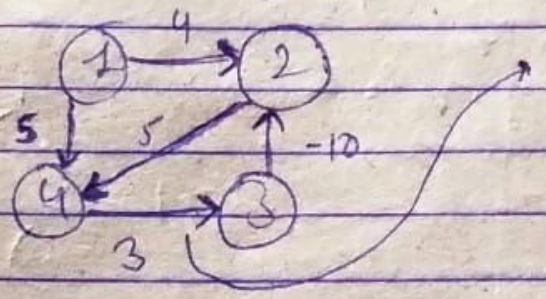
1) Relax all the edges (n-1) times

Relaxation →



[if (d[u] + wt < d[v])
d[v] = d[u] + wt

Mostly after (n-1) step the Relaxation of a graph is done but a drawback.



there is no end of edges and hence it will have to be Relaxed.

O(VE)

→ TC

→ check for negative cycle if after (n-1) iteration, Relaxation still happen then cycle