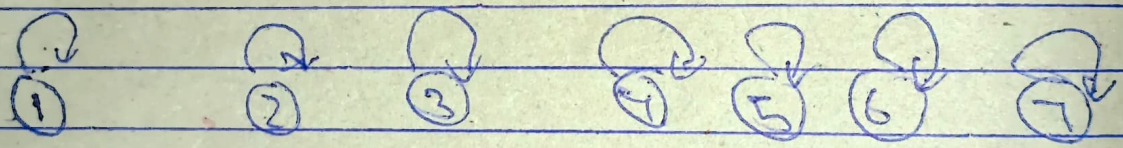
PAGE NO
DATE: / /

Disjoint Set

→ We can find whether two nodes are lying in same components of a graph or not

Two main oprat" find parent & find set() union() or unionset()

Union by Rank + path Compression



union (1,2)

↳ how to do union

→ get parent of 1 → 1

→ get parent of 2 → 2

→ compare rank of parents

→ If both have same rank kisi ko bhi kisi ke sath attach karlo.

find parent (1) - 1

2 - 2

3 - 3

4 - 4

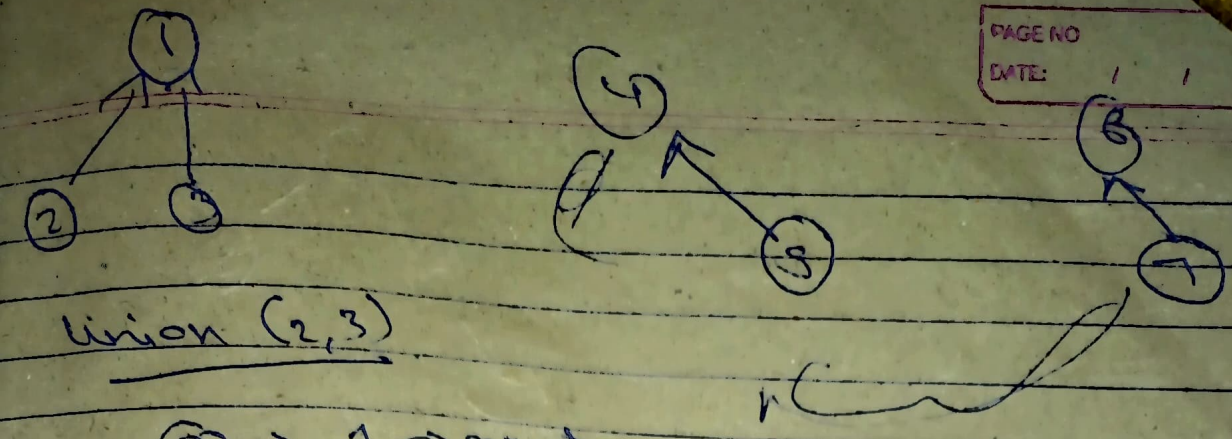
5 - 5

6 - 6

7 - 7

Rank 1 2 1

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



Union (2,3)

→ (2) → 1 → parent
→ (3) → 3

1 ki rank 1
3 ki rank 0
rank[3] < rank[1]

In this case parent[3] = 1

Union (4,5)

4 → [4] → parent
5 → [5] → parent

rank[4] = rank[5]

parent[5] = 4
rank[4] ++

Union (6,7)

6 → [6]
7 → [7]

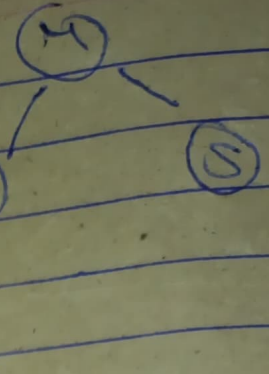
rank[6] = rank[7]

parent[7] = 6
rank[6] ++

PAGE NO
DATE : / /

Union [5, 6]

5 -> 4 -> parent
6 -> 6



rank [4] = rank [6] = 1

parent [6] = 4

rank [4] ++

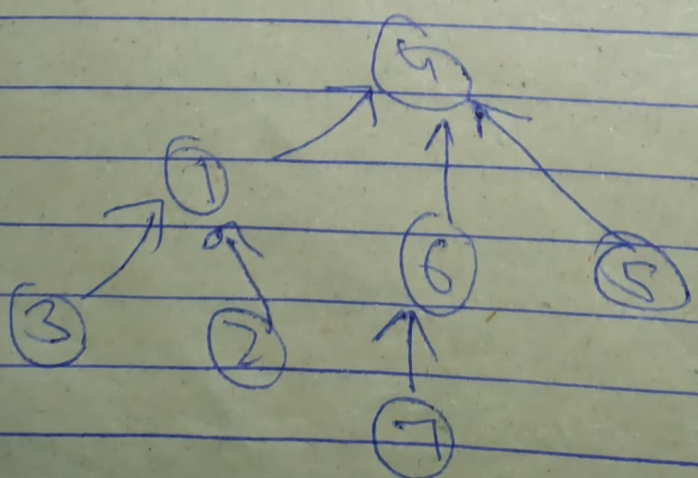
Union (3, 7)

3 -> 1 Rank 1 = 1

7 -> 4 Rank 4 -> 2

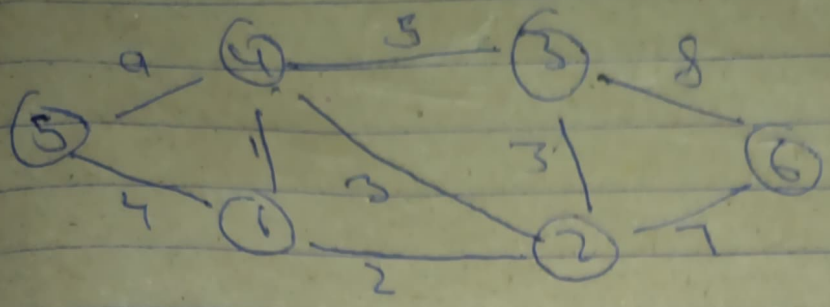
rank [1] < rank [4]
1 < 2

parent [1] = 4



PAGE NO.
DATE: / /

KRUSKAL'S ALGO



→ No need of adjacency ~~list~~ ✗

(2) We will be requiring a linear data structure that will store edges i.e 2 nodes & weight b/w them which will be sorted according to their weight.

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

Sorted wt

	u	v
1	1	4
2	1	2
3	2	3
3	2	4
4	1	5
5	3	4
6	2	6
8	3	6
9	4	5

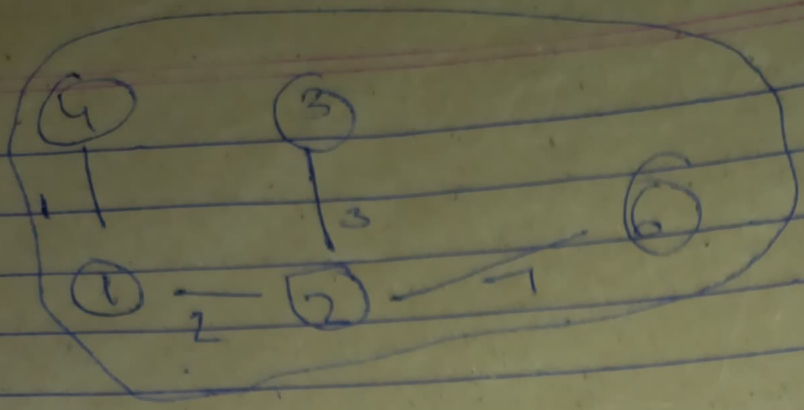
u v

1 4

1 → parent 1

4 → parent 4

If parent different then merge/union if same then do nothing zero different.



MST

1

not same comp ⇒ merged

2

2 & 3 → not same ⇒ merge

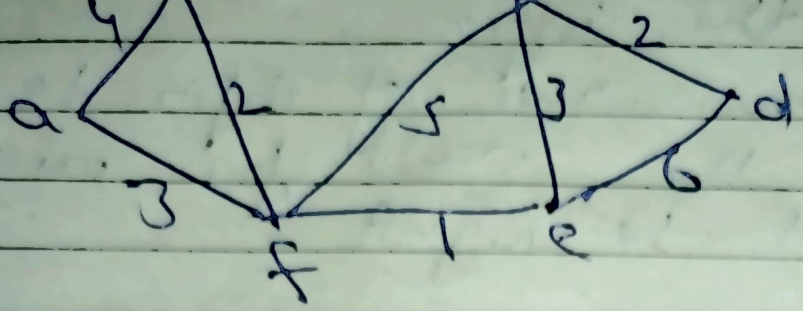
2 & 4 same comp ⇒ ignored

1 & 5 diff comp ⇒ merged

2 & 6 diff comp ⇒ merged

3 & 6 same comp ⇒ ignored

4 & 5 same comp ⇒ ignored.



a	b	c	d	e	f
0	∞	∞	∞	∞	∞
0	1	4	∞	∞	∞

a se b or c pe ja sakte hai to h unke distance mark ka di

then we took minimum of distance & fixed it & then repeated the if at some position we reach from a certain node & the distance is less than the value present in array then we do not do anything else update.

0	1	3	8	6	∞
0	1	3	8	4	∞
0	1	3	7	4	10
0	1	3	7	4	9

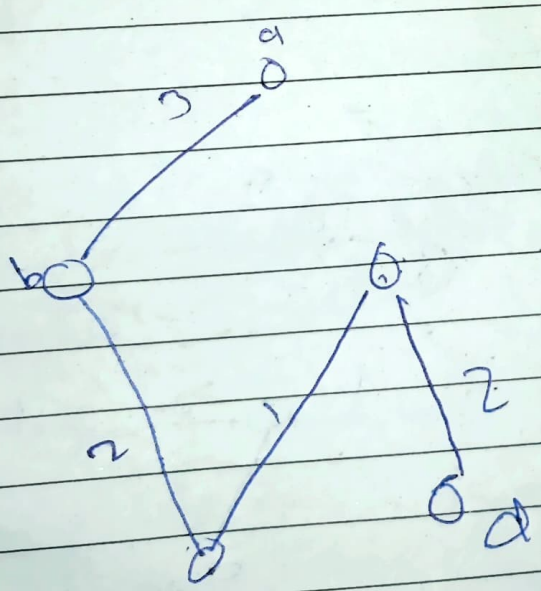
Page No.
Date

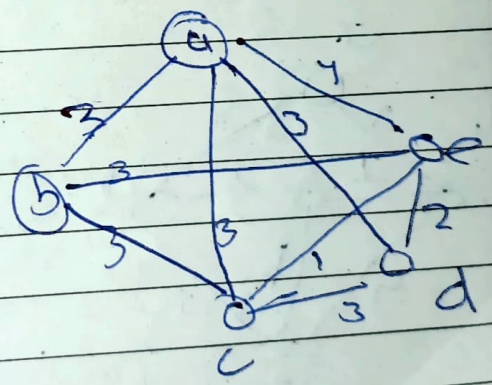
Prim's Algo

⇒ Select any vertex & choose the edge of smallest weight from it.

⇒ At each stage, choose the edge of smallest weight joining a vertex already included to vertex not yet included.

⇒ Continue until all vertices are includes.





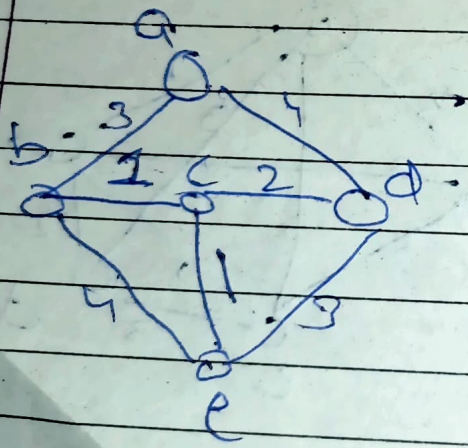
KRUSKAL'S ALGO

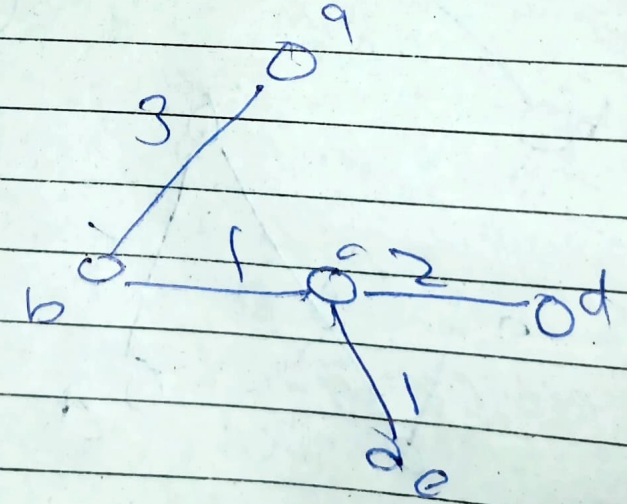
Working Rule

(i) Choose an edge of minimum weight.

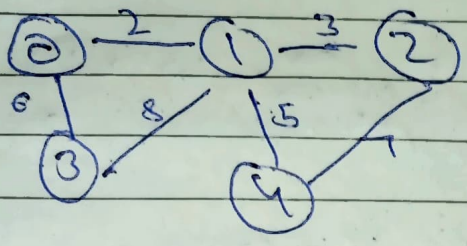
(ii) At each step choose the edge whose inclusion will not create a circuit

(iii) If G has n vertices stop after n-1 edges





Prim's Algo



Key

0	1	2	3	4
0	∞	∞	∞	∞
	2		6	

MST ⇒ To track MST mei kon kon hai

0	1	2	3	4	
F	F	F	F	F	

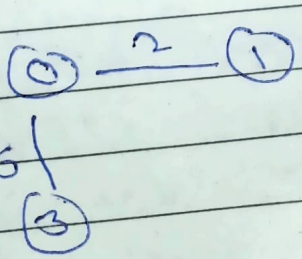
Parent

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

Shuruat mei sabka parent -1

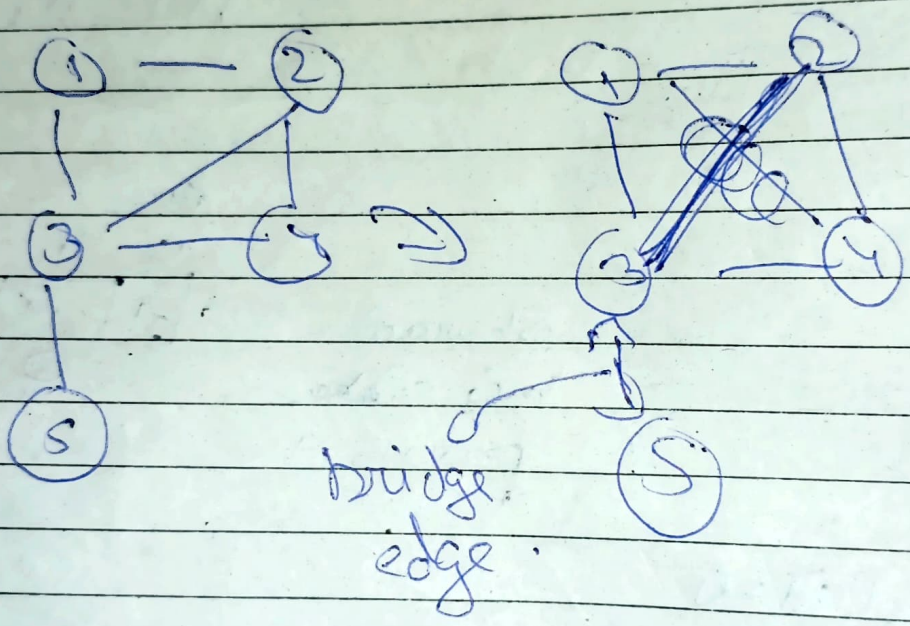
key[0] = 0
parent[0] = -1

minimum
u → 0
mst[u] = true

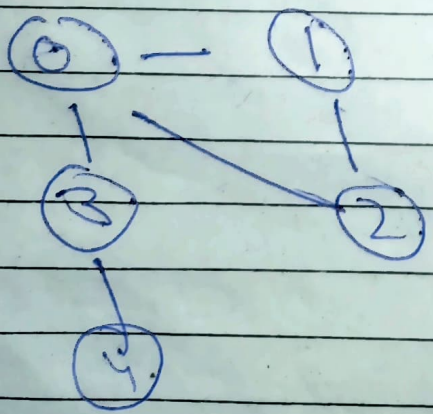


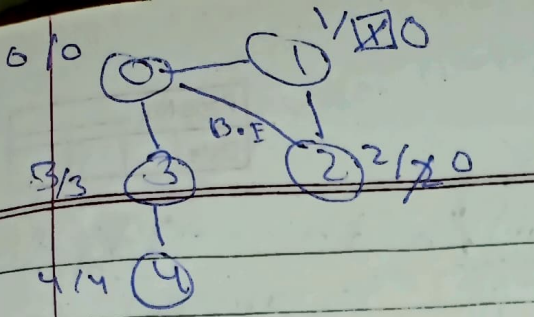
BRIDGE IN A GRAPH

Bridge is an edge on which no. of components ~~increase~~ remaining connected component increases.

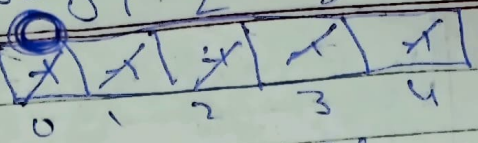


AIGO

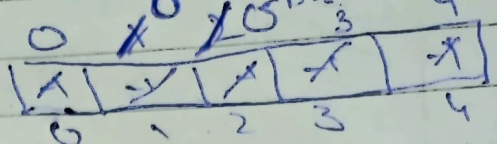




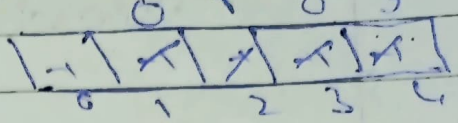
discovery 0 1 2 3 4



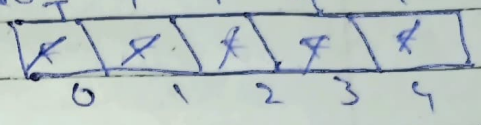
low 0 0 0 3 4



parent



visited T T T T T



timer = 0, 1, 2 3 4

---

discovery time is the time at which we found the node

low array is to store earliest possible time to reach that node the time taken to reach may vary from path to path.

Jab timer bhadda or hum 2 se 0 tak pahuchne ka try kiyya toh humne dekkha ki 0 already visited hai iska matlab (0)-(2) wali edge back edge hai matlab 2 tak kam time mei pahucha ja sakta hai toh 2 ka low time update karna padegga.

Back edge case

low[node] = min(low[node], disc[neighbour])

(disc[neighbour] = discovery time of neighbour)

If neighbour == parent ignore rakhte ta
warna ye back edge ho jayengi

In Back Edge Case

low[node] = min(low[node], disc[neighbour])

neighbour jiske sath back edge bani hai

While returning
low[node] = min(low[node], low[child])

To Check Bridge

low[neighbour] > disc[node]

→ Bridge present

for (0) (1) — (2)
neighbour = 1   node = 1

low[neighbour] = 0
disc[node] = 1

0 ≯ 1

∴ no Bridge present

This condition works because if

low[neighbour] > disc[node] it means there was only a single way through which we could have reached that node. Thus it is a Bridge edge.

ANUSUT

To Check A.P

++ parent != -1

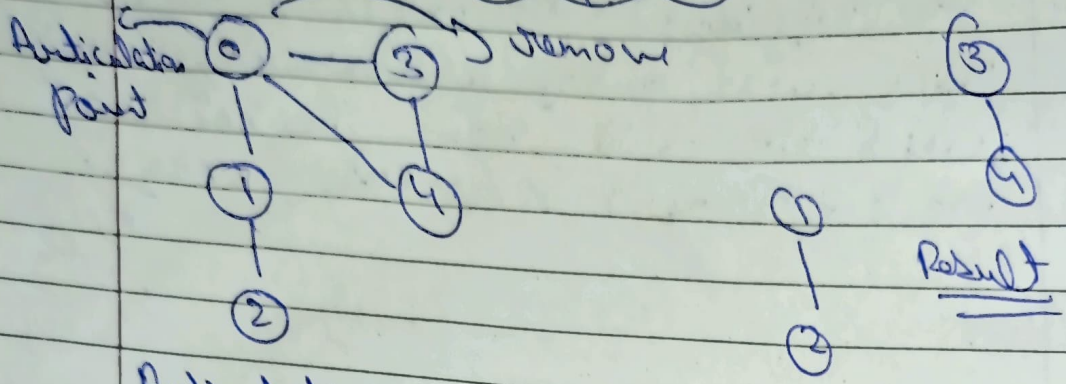
low[nbr] >= disc[node]

if nbr == parent ignore

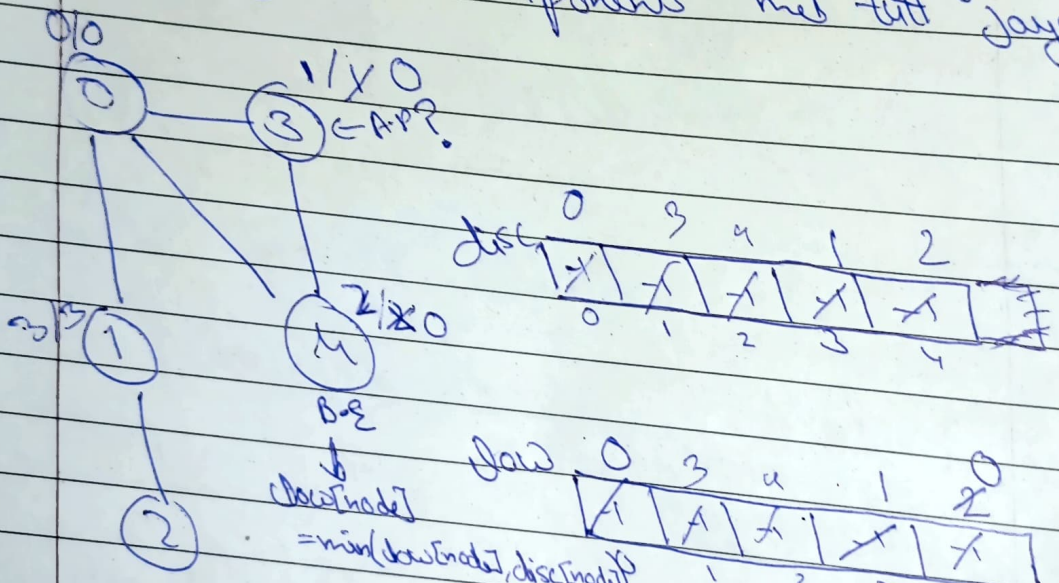
Page No.

Date

ARTICULATION POINTS IN A GRAPH



Articulation point is a node jisko agar remove kardo toh graph do ya do se jyada components mei tutt jayegg.



disc

0	3	4	1	2
0	1	2	3	4

low

0	3	4	1	0
1	2	3	4	

parent

0	1	0	3
0	1	2	3	4

visT

T	T	T	T	T
0	1	2	3	4

while returning low[node] = min(low[node], low[child])

Page No.
Date

Checking for 3 A.P

low[nbr] >= dis[node] && parent

for 3

0 >= 1 False

∴ 3 is not an A.P

This condition means that there is only one way to reach that particular node ∴ it will be distorted into components on removing that

for 0th node

low[nbr] ≥ dis[node] && parent != -1

4 ≥ 3

Yes ∴ it is an AP

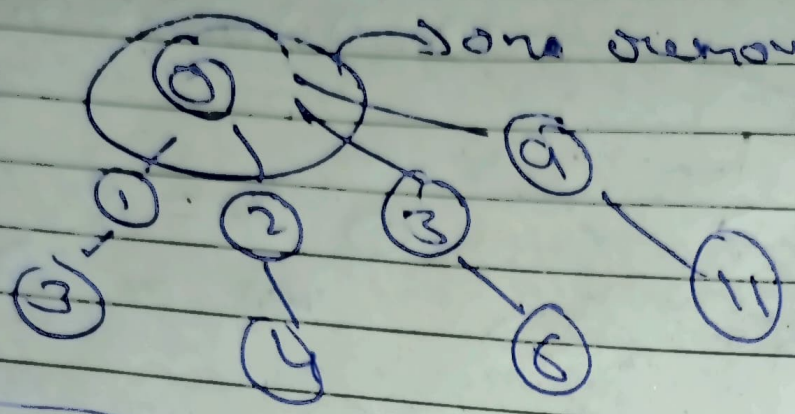
for 0 3 ≥ 0 True but parent = -1

∴ false

But it was an A.P therefore we need to handle it separately.

Page No.
Date

parent != -1 → reason → some removal in component

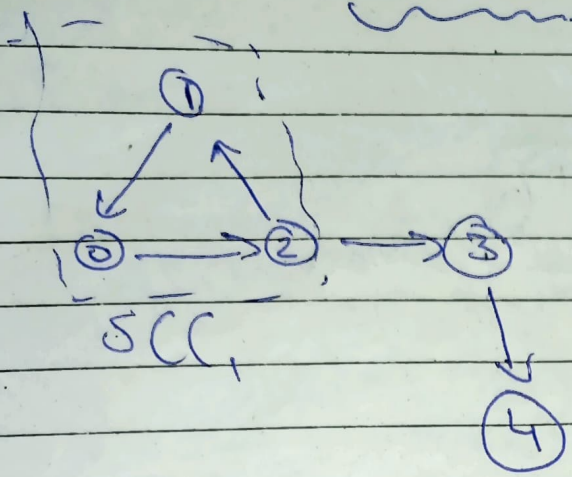


suppose

[1] parent == -1 → child is articulation point

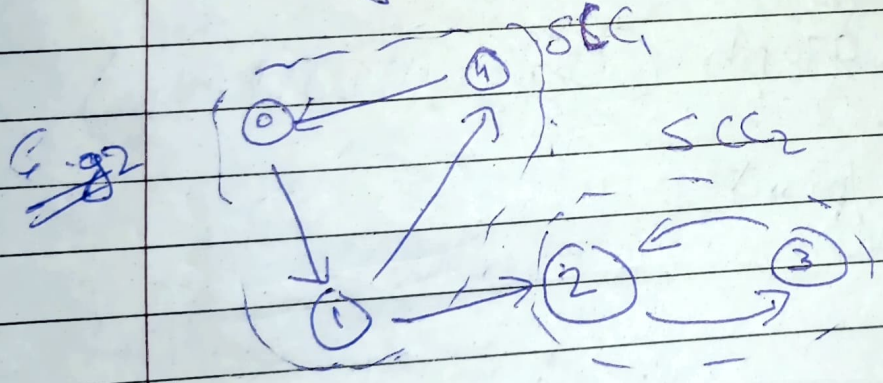
return true

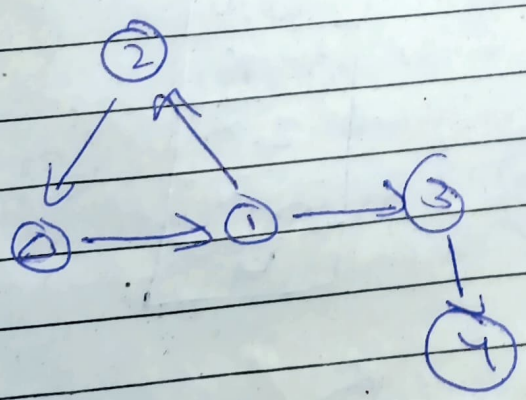
Kosaraju's AlGo

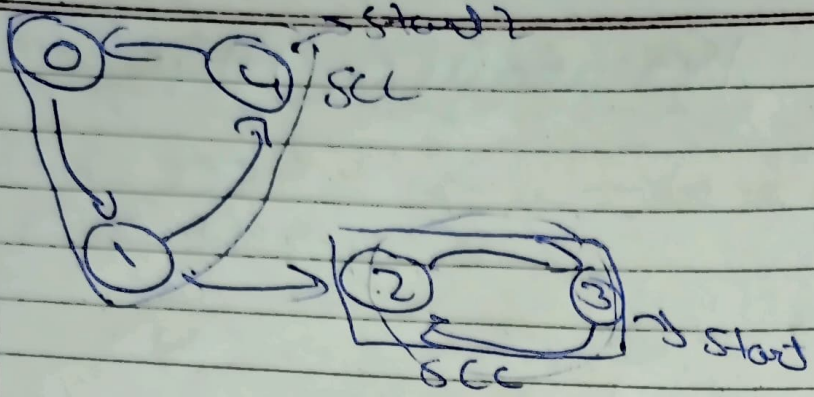


SCC :-
-> 0 2 1
-> 3
-> 4

Strongly connected component is that in which we can traverse the entire component taking any node as the starting point.





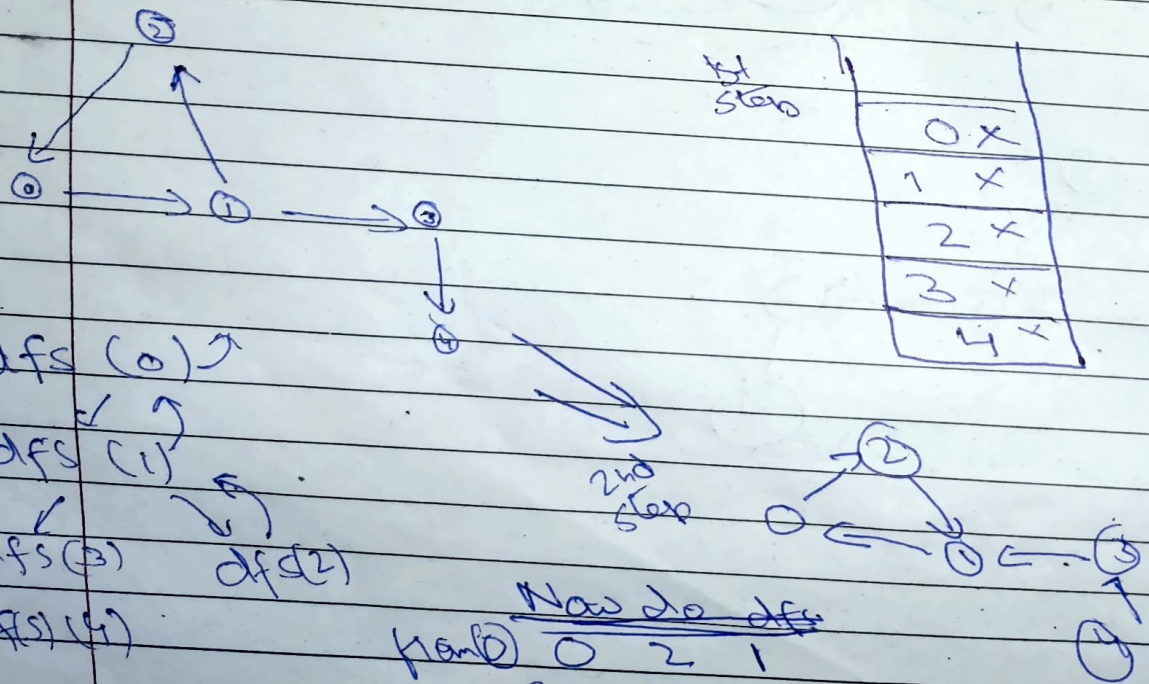


Kosaraju algo

⇒ Sort all nodes basis on their finishing time
↓
Topological Sort

⇒ Transpose graph (change direction of edge)

⇒ dfs → count / print SCC



Page No.
Date

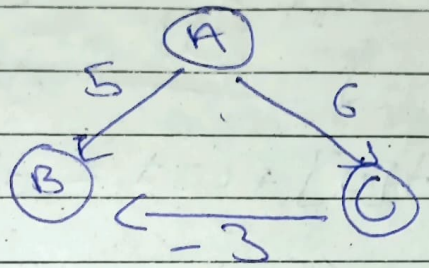
Bellmanford Algo

Shortest Path finding Algo

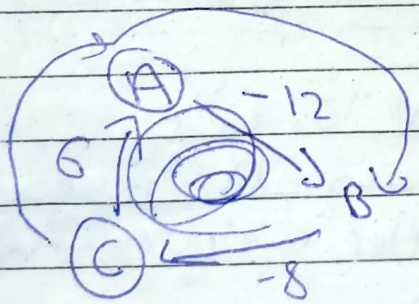
Dijkstra's Algo doesn't work for -ve weights.

↓

Suc greedy



for bellman ford -ve cycle should not be present in graph.



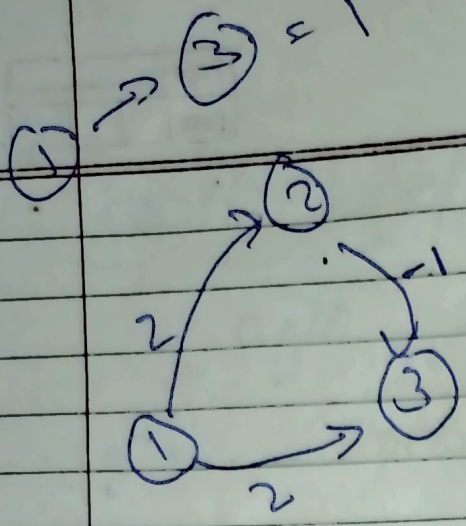
$6 + (-12) = -6 - 8 = -14$

$+6 - 8 - 12$

$= -20$

We can find -ve cycle within a graph.

Bellman → for directed weighted graph.



We have to apply the same given formula (n-1) times for all edges

Step I

```
if (dist[u] + wt < dist[v])
    dist[v] = dist[u] + wt
```

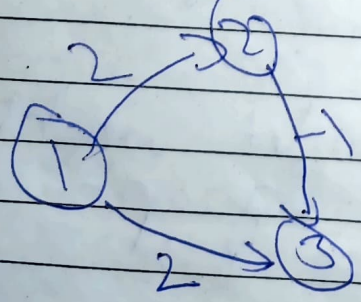
Step II

1 more time → same formula

if any distance gets updated then it means -ve cycle is present.

n=3

2 times formula apply



edges

1 → 2 (2)

2 → 3 (-1)

1 → 3 (2)

0	∞	∞
1	2	3

2 2 1