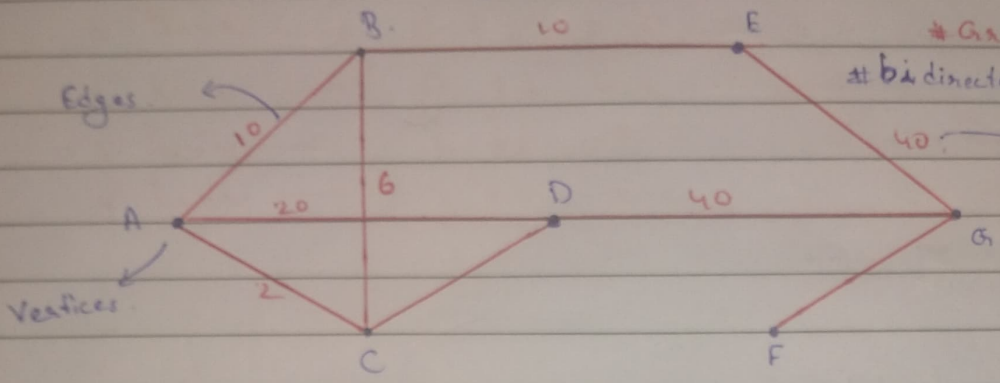
Graphs.

→ Dist / Cost / Time

Edges, Vertices, Weight, Direction.

Google Maps

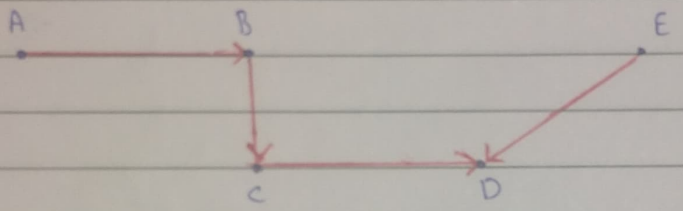
A ⟶ B. Shortest Dist / Shortest Time



Graph

bi directed.

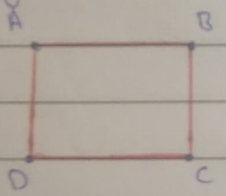
Directed Graph [One way Roads]



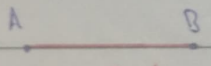
⇒ [D → E & B → A Not Possible]

⇒ [A → D & E → D Possible]

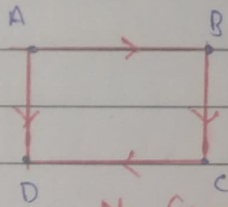
Cycle → Loop + you should be able to reach back to st position without moving back



Cycle ⇒ A → B → C → D



No Cycle

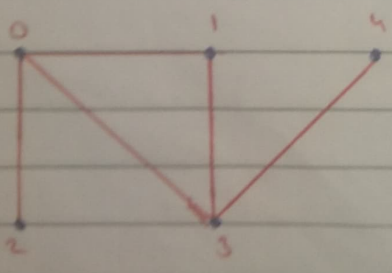


Directed

No Cycle

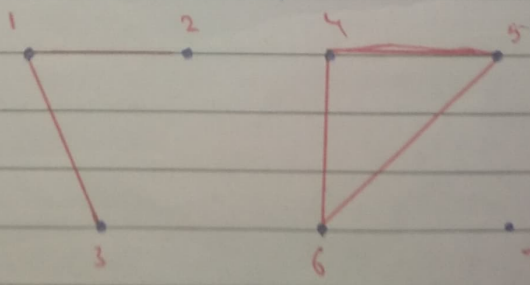
Components

Graph



1 Component

Graph



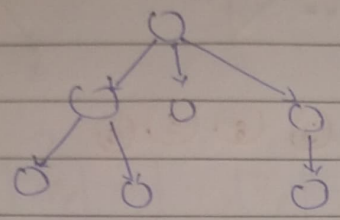
3 Component

1, 2, 3 → 4, 5, 6 → 7.

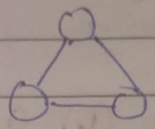
Representation of Graph:-

Any tree is a graph.

But Any graph may or may not be a tree



Tree & # Graph also



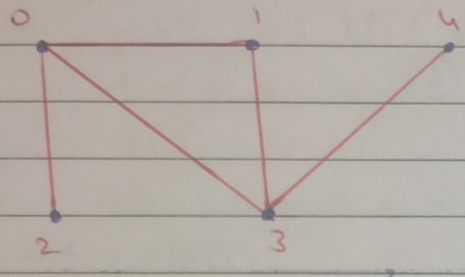
Graph
Not Tree.

=> Representation.

↳ we have 2 ways :- ① Adjacency Matrix → 0's, 1's (can be values as well.)

↳ ② Adjacency List → to save space

For ex :-



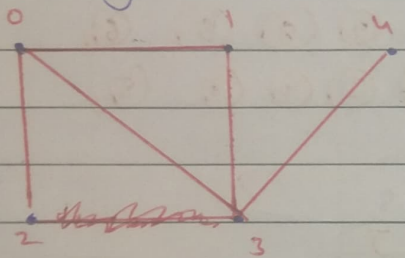
	0	1	2	3	4
0	✗	1	1	1	0
1	1	✗	0	1	0
2	1	0	✗	0	0
3	1	1	0	✗	1
4	0	0	0	1	✗

0 is Not Directly Connected to 4

3 is directly Connected to 4

=> Adjacency List.

List < List < Integer >> list ;



0 : {1, 2, 3}

1 : {0, 3}

2 : {0}

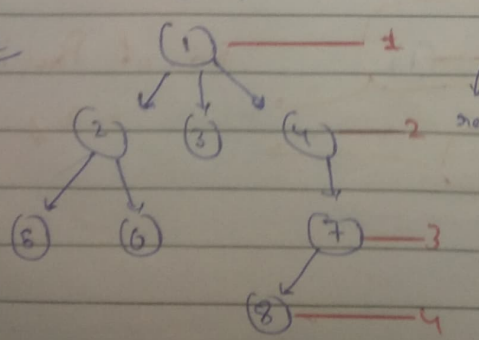
3 : {0, 1, 4}

4 : {3}

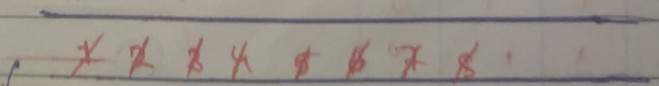
adj = { {0, 1, 3}, {0, 3}, {0}, {0, 1, 4}, {3} }
0 1 2 3 4

Breadth First Search (Queue)
(BFS)

Tree

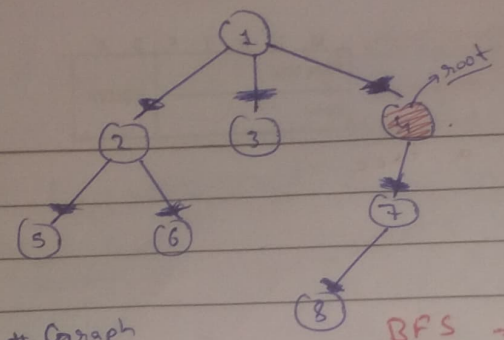


front rear



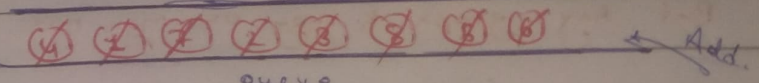
remove que Add

→ 1 2 3 4 5 6 7 8.



Need to mark Visited.

front



rear

remove

queue

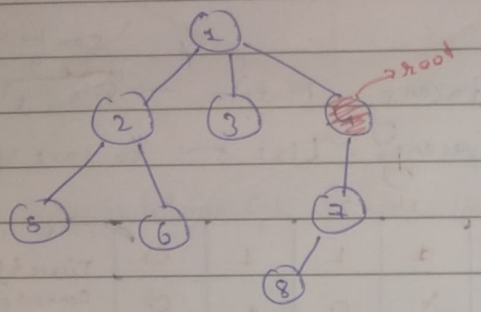
Add.

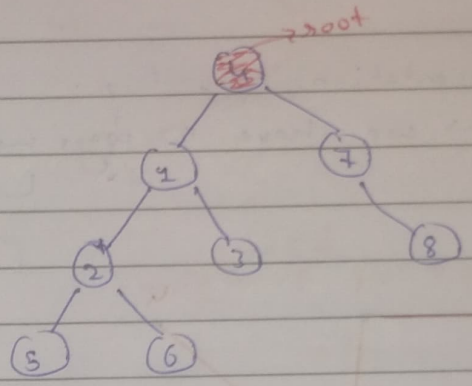
Graph

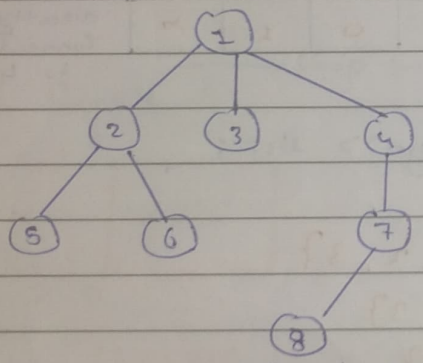
BFS ⇒ 4 1 7 2 3 8 5 6

About 4

Visualisation:-

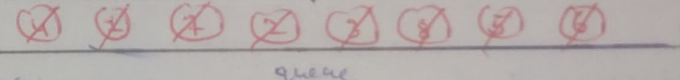






front

rear



remove

queue

Add.

4 → 4 1 7 2 3 8 5 6

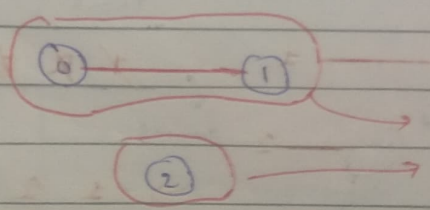
6 → 6 2 1 5 3 4 7 8

0	1	2	3	4	5	6	7	8
visited ⇒	T	T	T	T	T	T	T	T	T

---

① LeetCode Q.No (547) { Number of Provinces (Bfs) } → Hint Bfs

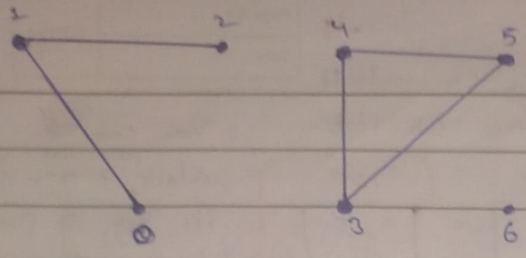
	0	1	2
0	1	1	0
1	1	1	0
2	0	0	1



No. of Provinces = 2

Graph

Count = 0 1 2 3.




```
for (i=0 to 6)
    if(!vis[i]){
        bfs(i)
        count++
    }
}
```

M	T	W	T	F	S	S
Page No.:						YOUVA
Date:						

front rear

0 1 2 3 4 5 6

vis = [T F | T F | T F | T F | T F | T F | T F]

Remove

que

Add

3 bfs kage.

0 1 2 | 3 4 5 | 6

1st bfs | 2nd bfs | 3rd bfs

Worst Case :- $O(n^2)$ TC — Adj Matrix

Every Case ⇒ → Adj. List

$O(V + 2E)$ — E: Edges, V: vertices

=> Code :-

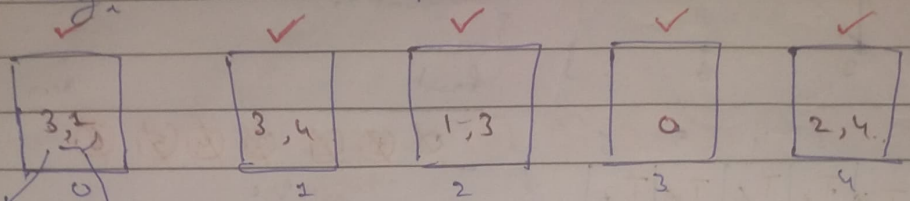
```
pubic void bfs(int i, boolean[] vis, int[][] adj){
    int n = adj.length;
    vis[i] = true;
    Queue<Integer> q = new LinkedList<>();
    q.add(i);
    while(q.size()>0){
        int front = q.remove();
        for(int j=0; j<n; j++){
            if(adj[front][j]==1 && vis[j]==false){
                q.add(j);
                vis[j] = true;
            }
        }
    }
}
```

```
for(int i=0; i<n; i++){
    if(!vis[i]){
        dfs(i, vis, adj);
        count++;
    }
}
```

② LeetCode Q.No. (841) { Keys And Rooms (BFS) }

'n' rooms → 0 to n-1. ⇒ Check if you can unlock All rooms or Not.

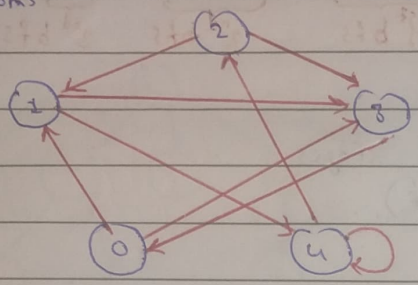
⇒ Initially only room '0' is opened.

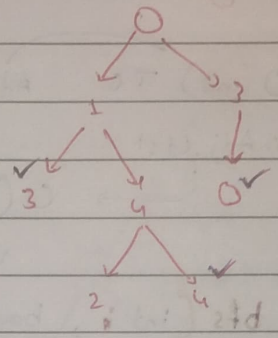


Keys of another Rooms

You can only visit to these two rooms from '0' room.

You can only unlock particular room if its key is present in the room in which you currently are in.





Bfs:

adj = { {1,3}, {3,0,1}, {2}, {0} }

vis = | T | T | F | T |
0 1 2 3

front rear

q: 0 1 3

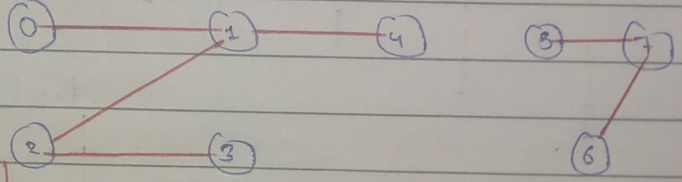
remove ← Add

You are Not Able to Visit All the rooms so return false

③ LeetCode Q.No. (1971) { Find if Path Exists in Graph (BFS) }

(start, end)

Need to check if we can go from Start to End



vis = | T | T | T | T | T | F | F |
0 1 2 3 4 5 6
st end

4 → 3 Yes.

2 → 5 No.

edges = { {0,1}, {4,1}, {1,2}, {2,3}, {7,5}, {6,7} }

Represents you can go from 0 to 1 & [1 to 0].

A.S ⇒ O(V + 2E) or O(n+2E)

T.C ⇒ O(n+2E)

adj = { {}, {}, {}, {}, {}, {}, {}, {} }

0 :- 1
1 :- 0, 4, 2
2 :- 1, 3
3 :- 2
4 :- 1
5 :- 7
6 :- 7
7 :- 5, 6.

(4) LeetCode Q.No. (200) → Number of Islands (BFS) → Connected Components

Page No.: Date: YOUVA

Island

Ans => (1)

Island Island Island

Ans = (3)

Dry Run (1)

1st BFS

2nd BFS

3rd BFS

T F	T F	F	F	F
T F	T F	F	F	F
F	F	T F	F	F
F	F	F	T F	T F

Vis

Count = ∅ ∅ ∅ (3)

Ans

Remove

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

1st BFS | 2nd BFS | 3rd BFS

Dry Run (2)

1st BFS

T F	T F	T F	T F	F
T F	T F	F	T F	F
T F	T F	F	F	F
F	F	F	F	F

Vis

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

Remove

Add

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

1st BFS

Ans

Count = ∅ (1)

(x,c-1),c

x,c → x,(c+1)

(x+1),c

1	1	1
0	1	0
1	1	1

T	T	T
F	T	F
F	T	T

Vis

* A.S => O(n*n)
* T.C => O(n*n)

You have to Apply BFS for All direction Top, Bottom, left, right.

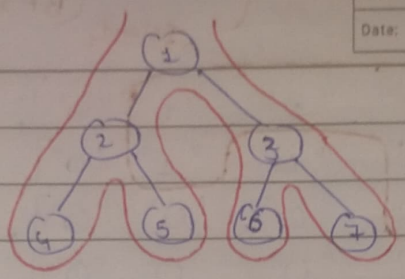
Dfs :- Depth First Search. → keep Traversing down.

Binary Tree → Pre, In, Post

→ N L R

→ L N R

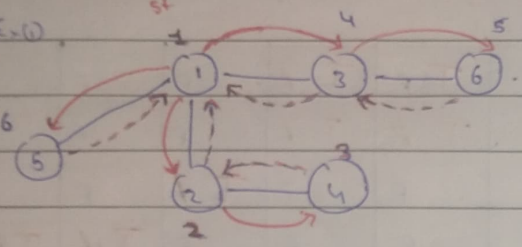
→ L R N.



Euler Tree

Number of Provinces (Dfs)

Ex①



1 → 2, 3, 5

2 → 1, 4

3 → 1, 6

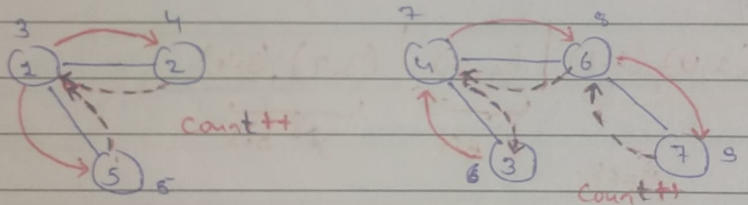
4 → 2

5 → 1

6 → 3.

vis = | 1 | 2 | 3 | 4 | 5 | 6 |
| T/F | T/F | T/F | T/F | T/F | T/F |

Ex②



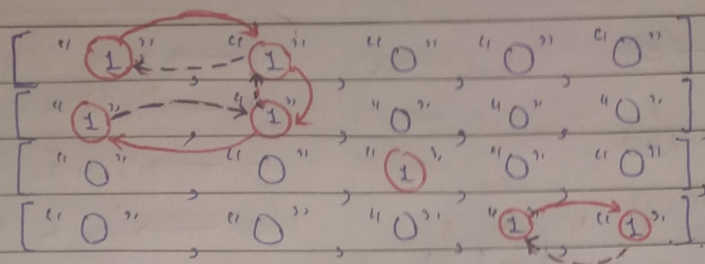
vis = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| T/F | T/F | T/F | T/F | T/F | T/F | T/F | T/F | T/F |

=> Code :-

```
public void dfs(int i, boolean[] vis, int[][] adj){
    int n = adj.length;
    vis[i] = true;
    for(int j = 0; j < n; j++){
        if(adj[i][j] == 1 && vis[j] == false){
            dfs(j, vis, adj);
        }
    }
}
```

Number of Islands (DFS) :-

LTRB



T	T	F	F	F
T	T	F	F	F
F	F	T	F	F
F	F	F	T	T

vis

count = ~~0~~ ~~1~~ ~~2~~ 3 → Ans.

=> Code :-

```
p v dfs(int i (row), int j (col), char[][] grid, boolean[][] vis){
    int m = grid.length, n = grid[0].length;
    vis[i][j] = true;
    if(i-1>=0 && grid[i-1][j]=='1' && vis[i-1][j]==false)
        dfs(i-1, j, grid, vis);
    if(i+1<m && grid[i+1][j]=='1' && vis[i+1][j]==false)
        dfs(i+1, j, grid, vis);
    if(j-1>=0 && grid[i][j-1]=='1' && vis[i][j-1]==false)
        dfs(i, j-1, grid, vis);
    if(j+1<n && grid[i][j+1]=='1' && vis[i][j+1]==false)
        dfs(i, j+1, grid, vis);
}
```

Keys & Room (DFS) :-

=> Code :-

```
p v dfs(int start, boolean[] vis, List<List<Integer>> adj){
    vis[start] = true;
    for(int ele : adj.get(start)){
        if(!vis[ele]){
            dfs(ele, vis, adj);
        }
    }
}
```

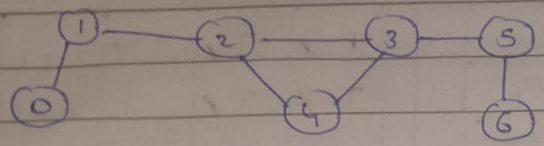
~~=> Code :- p v dfs(i, vis, adj){~~
~~vis[i] = true;~~

H.W => Find if Path Exists in Graph ,, → Same Code (DFS)

Cycle Detection:-

queue<node, parent>

① ⇒ In Undirected Graph (Bfs).



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

Visited = | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| T | T | T | T | T | T | F |

Node, Parent

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

remove — q — Add

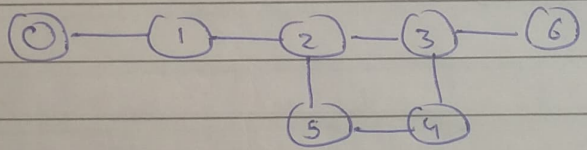
(0, -1), (1, 0). (2, 1). (3, 2).

We will Not insert this parent Again.

It can insert (4) but it is Alread visited & it is also Not a Parent of (3) so Cycle detected.

Cycle Detected.

② ⇒ In Undirected Graph (Dfs)



vis = | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| T | T | T | T | T | T | F |

Node, Parent

dfs(0, -1).

dfs(1, 0).

dfs(2, 1)

dfs(3, 2). dfs(5, 2)

dfs(4, 3) dfs(6, 3).

dfs(5, 4). Return True

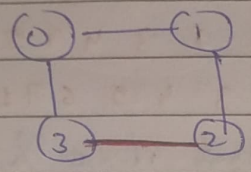
It Can Call (4), But (4) is his Parent. So Not Possible

It Can Call (2) Also it is alread visited & Not a parent of (5) so Cycle Detected

(5) LeetCode Q.No (785) { Is Graph Bipartile. }

M T W T F S S | Page No.: | Date: | YOUVA

Ex (1):-



0 → 1,3
1 → 0,2
2 → 1,3
3 → 0,2

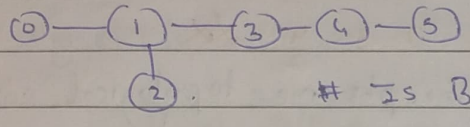
Set A	Set B
0, 2	1, 3

True → Bipartile

(1) Nodes in a Particular Set must Not Connects each other Directly.

(2) → Every Two Nodes forming An edge must be from different different sets.

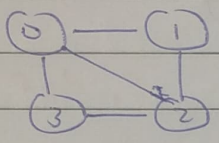
Ex (2):-



Set A	Set B
0, 2, 3, 5	1, 4

Is Bipartile.

Ex (3):-



False

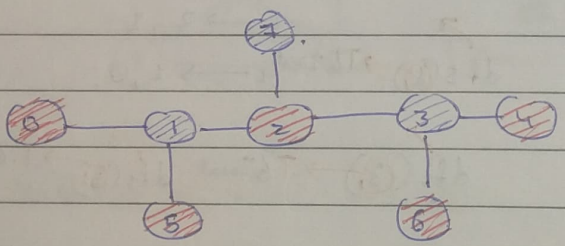
↳ Not an Bipartile

Set A	Set B
1, 3	0

You are Not Able to place (2).

Observation :-

When we colour each node of the graph with either red or blue & Adjacent Nodes must Have different Colour.

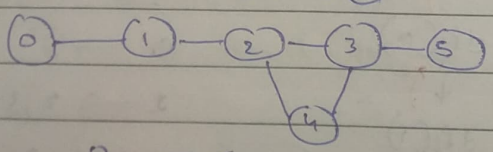


→ Any Acyclic Graph is Always Bipartile

→ Any Cyclic Graph with Even cyclic length is Bipartile

Dry Run :-

Red => 0
Blue => 1



0 → 1
1 → 0, 2
2 → 1, 3, 4
3 → 2, 5, 4
4 → 2, 3
5 → 3.

0	1	2	3	4	5
0 ✓	1 -1	0 -1	1 -1	1 -1	0 -1

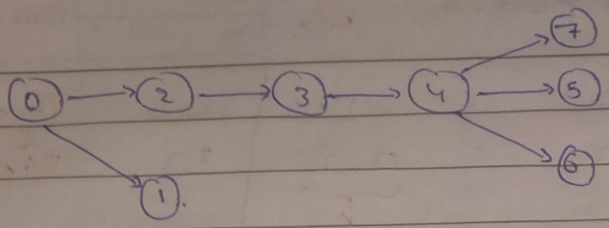
vis

Add → 0 1 2 3 4 5

Remove ← 0, 1, 2, 3, 4

So, Not Biphatile ← It can insert (4), (5). But (4) is Already Visited. And its Colour is same like (3).

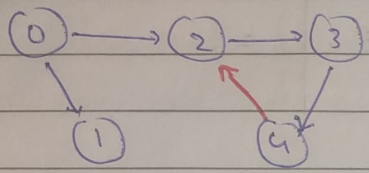
Topological Sort :- → DAG [Directed Acyclic Graph]



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

⇒ Ordering of All Nodes of the Graph such that if a & b are nodes & a → b then in the ordering 'a' will always come before 'b'.

⇒ If there is An edge b/w a & b then 'a' should Appear before 'b'.



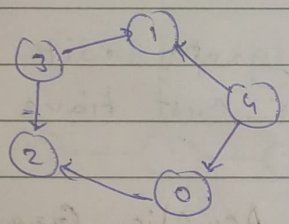
→ 0 1 2 3 4 ✗✗

⇒ Cyclic Graphs me topological sort Nahi Hoga.

DAG

⇒ DFS.

⇒ 3 4 1 0 2.



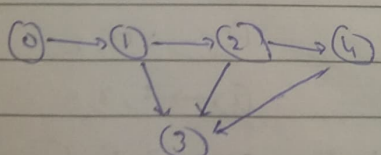
Adj
0 → 2
1 →
2 →
3 → 2, 1
4 → 1, 0

dfs(0) → T & Insert in Ans
dfs(2) → T & insert in Ans.
dfs(1) → T & Insert in Ans
dfs(3) → T & Insert
dfs(4) → T & Insert in Ans

vis = | T | T | T | T | T |
0 1 2 3 4

Ans = {2, 0, 1, 3, 4}
Reverse {4, 3, 1, 0, 2}

⇒ DFS.



vis = | T | T | T | T | T |
0 1 2 3 4

Ans = {3, 4, 2, 1, 0}
Reverse {0, 1, 2, 4, 3}

dfs(0)
dfs(1)
dfs(2)
dfs(3)
dfs(4)
dfs(3)
dfs(3)

Adj List
0 → 1
1 → 2, 3
2 → 4, 3
3 →
4 → 3

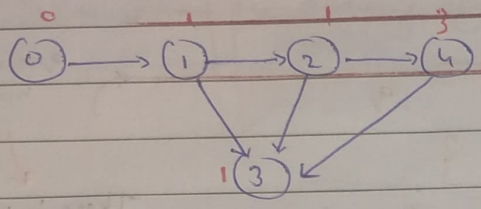
```
for(i=0 to n-1){
    if(!vis[i])
        dfs(i);
}
```

```
Void dfs(i){
    vis(i) = true;
    for(int ele : adj[i]){
        dfs(ele);
    }
    ans.add(i);
}
```

M T W T F S S | Page No.: | Date: | YOUVA

⇒ Bfs (Kahn's Algorithm) :-

Ex:-



Adj. List

0 → 1

1 → 2, 3

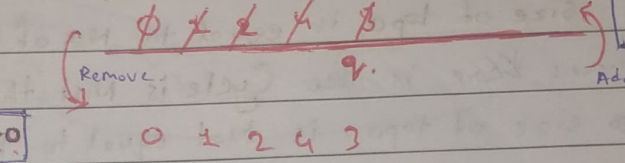
2 → 3, 4

3 →

4 → 3

In Degree & Out Degree

	0	1	2	3	4
in =	0	~~1~~ 0	~~1~~ 0	~~2~~ 0	~~1~~ 0



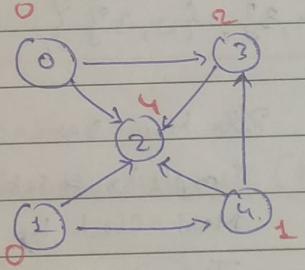
0 1 2 4 3

ans = { 0, 1, 2, 4, 3 }

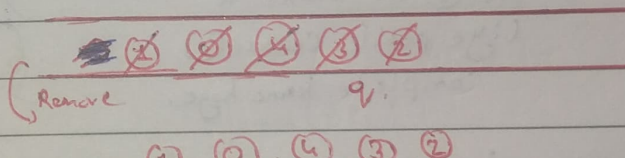
Starting mai jinki indegree zero hai unko he insert karo

Indegree zero hogi tabhi hum queue mai ele insert karenge.

Ex:-



⇒ 0, 1, 4, 3, 2 or 1, 4, 0, 3, 2.



ans = { 1, 0, 4, 3, 2 }

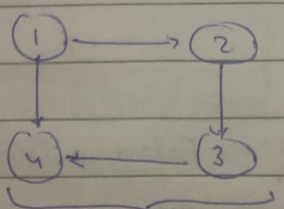
1, 0, 4, 3, 2

	0	1	2	3	4
in =	0	0	~~2~~ 0 ~~1~~	~~2~~ 0	~~1~~ 0

Cycle Detection :-

⇒ Directed Graph (BFS) :-

(BFS) says There is a Cycle. → ~~false~~ This will Not work.



Cycle is Not there

(1, -1) (2, 1) (4, 1) (3, 2)

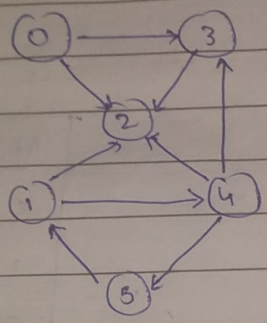
vis =	①	②	③	④
	T ~~F~~	T ~~F~~	T ~~F~~	T ~~F~~

it can insert ~~(4)~~ but it is already visite & it is ~~not~~ a parent of (3) so, Cycle is There

=> In DAG, When we Apply topological Sort → BFs (kahn's) we visit "all" the Nodes.

=> So, if we have A Directed Cyclic Graph & we Apply kahn's Algorithm then something unusual happens.

Ex:



	0	1	2	3	4	5
in =	0	1	~~2~~ 3	~~2~~ 1	1	1

	0	1	2	3	4	5
vis =	T	F	F	F	F	F

queue khali ho gya

There is No ele left with zero indegree

topo = { 0 }

Observation! -

=> So, if size of topo is equal to No. of Nodes in Graph Cycle is Not there.

=> if size of topo is Not equal to No. of Nodes in Graph then Cycle is there.

#6 LeetCode Q. No. 207 { Course Schedule }

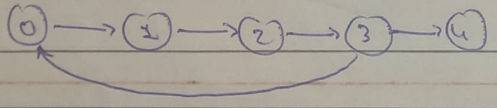
pre = { {3, 2}, {1, 0}, {2, 1}, {4, 3}, {0, 3} }

Courses

Course 3 ko karne k liye pehle Course 2 complete karna hoga.

Hum sare course complete kr sakte hai ki Nahi Check kr na hai. (T/F).

=> In this directed graph, if it is cyclic → True, else False



	0	1	2	3	4
in =	1	1	1	1	1

Kahn's Algorithm (BFs) work nhi karega.

Adj - List

0 → 1 } → Course 1 ko karne k liye 0 ki Req hai

1 → 2

2 → 3

3 → 0, 4

4 → X.

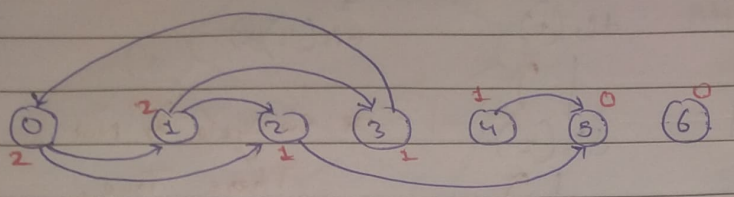
LeetCode Q. No. 802 { Course Schedule II } => Follow up Question.

=> easy peasy.

- Need to return order of Course we need to follow to complete All Courses.
- Return Topological Sort.

(7) LeetCode Q.No. 802 → Find Eventual Safe states.

↳ Need to Return Safe Nodes → Find All the Nodes which are Not part of A Cycle



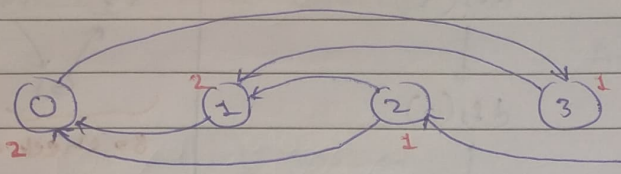
⇒ Terminal Node :- If there is No Out Going Edges Node is TN.

⇒ Saf Node :- If any Node's Path end's to TN then that Node is Safe Node

indegree → incoming edges.

terminal Node → No outgoing edges → Outdegree → 0

Reverse the Connections. Then → Kahn's Algo.



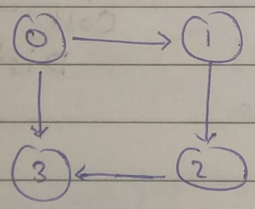
in = | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 |

Ans = {5, 6, 4, 2}

Remove ← q → Add

Cycle Detection :-

⇒ Directed Graph (DFS) :-



vis = | T | T | T | T |
| 0 | 1 | 2 | 3 |

0 → 1 → 2 → 3
&
0 → 3.
} Are diff. paths.

Normal (DFS)

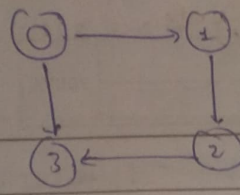
dfs(0,-1)

dfs(1,0) dfs(3,0)

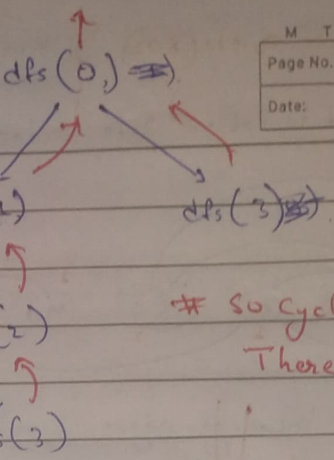
dfs(2,1)

dfs(3,2)

Cycle detected was But Cycle is Not There



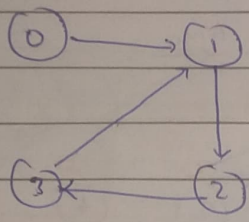
We don't Need to pass parents



vis = T F T F T F T F

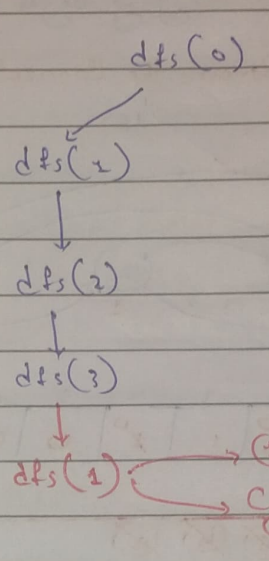
path = F F F F

So Cycle was Not There True

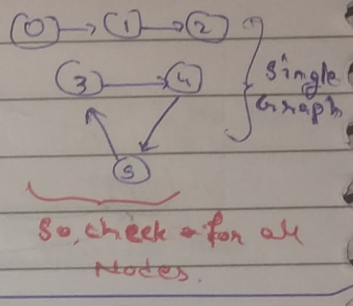


vis = T F T F T F T F

path = T F T F T F T F



Edge Case



So, check for all Nodes.

Dijkstra's Algorithm:-

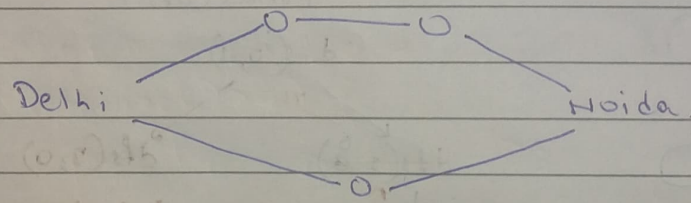
Undirected, Weighted

→ Shortest / longest Path.

Locations

↓

Cost, Time, Distance

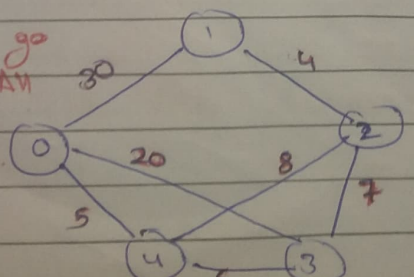


→ Least Cost → via metro → 2hr 100 Rs

→ Least Time → Via Cab → 1hr 800 Rs

Avg. Cost → Thodi si Cab, thodi si metro 1.5hrs 300 Rs

we have to go from 0 to All other in Min^n Cost



dist = 0 17 13 11 5

0 se leke

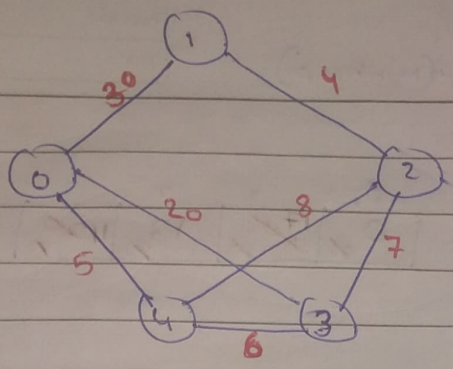
0→1 → 30, 5+8+4=17, 5+6+7+4→22

0→4 → 5, 20+6

0→2 → 20+7, 30+4, 5+8→13

0→3 → 20, 5+6→11

Dijkstra's Algo ⟶ BFS ⟶ Queue ✗
↳ PQ [Heap]

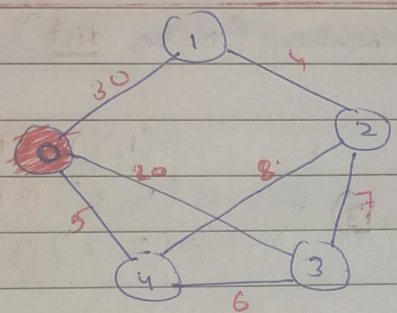


Adj = 0 ⟶ (1,30), (4,5), (3,20)
1 ⟶ (0,30), (2,4)
2 ⟶ (1,4), (3,7), (4,8)
3 ⟶ (0,20), (2,7), (4,6)
4 ⟶ (0,5), (2,8), (3,6)

List < List < Pairs >> Adj ;

⇒ Dijkstra's Algo :-
↳ BFS ⟹ Queue.. → Thoda sa Alag hai

Adj = 0 ⟶ (1,30), (4,5), (3,20)
1 ⟶ (0,30), (2,4)
2 ⟶ (1,4), (3,7), (4,8)
3 ⟶ (0,20), (2,7), (4,6)
4 ⟶ (0,5), (2,8), (3,6)
nodes ⇒ n , edges ⇒ ≈ n²
T.C ⇒ O(n²)



dist = | 0 | 17 | 13 | 11 | 5 |

(0,0), (1,30), (3,20), (4,5), (2,34), (2,27), (2,13), (3,11), (1,17)

queue< Node, distance >

Remove

(0,0) → (1,30) 0+30 ✗ ; (3,20) 0+20 ✓ ; (4,5) 0+5 ✓

(4,5) → (0,5) (10) ✗ ; (2,8) (13) ✓ ; (3,6) (11) ✓

(2,13) → (1,4) (17) ✓ ; (3,7) (20) ✗ ; (4,8) (21) ✗
dist[2] >= 13 we will check

(1,30) → (0,30) 30+30 ✗ ; (2,4) 30+4 ✓

(2,34) → (1,4) (38) ✗ ; (4,8) (41) ✗ ; (3,7) (42) ✗
dist[2] < 34 then we will Not Check

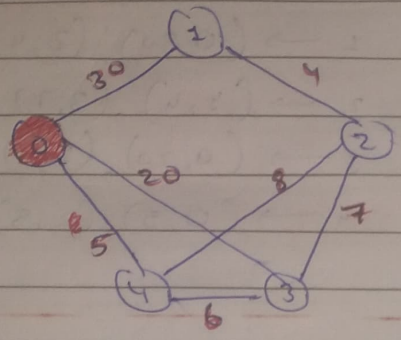
(3,11) → (0,20) (31) ✗ ; (2,7) (18) ✗ ; (4,6) (17) ✗

(3,20) → (2,7) (27) ✓ Add ; (4,6) (26) ✗ ; (0,20) (40) ✗
dist[2] < 27

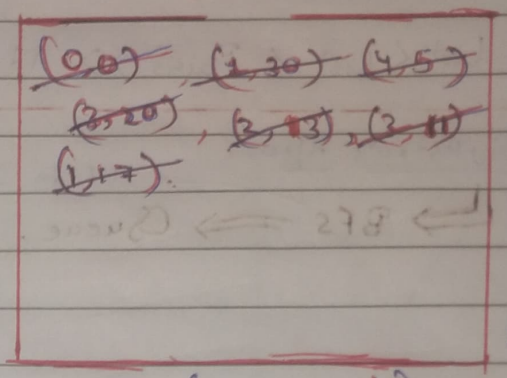
(1,17) → (0,30) (47) ✗ ; (2,4) (21) ✗

⇒ Dijkstra's Algo : → Thoda sa Alag.

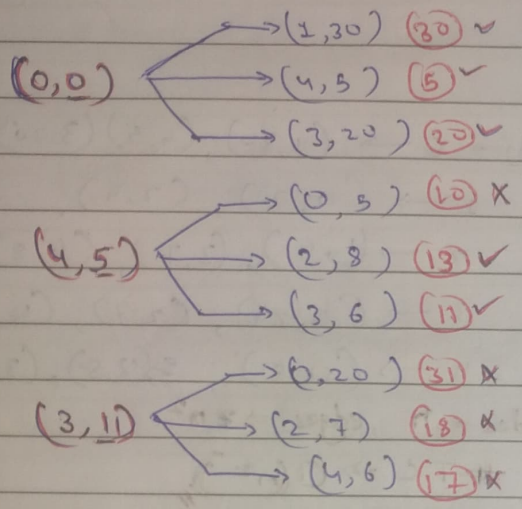
↳ ~~BFS~~ BFS ⟹ Heap. (MinHeap)

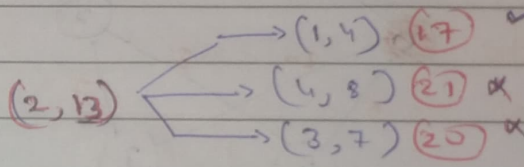


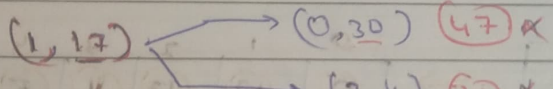
Cost = | 0 | 17 | 13 | 11 | 5 |



minheap (Node, dist)







(3,20) ✗ (1,30) ✗

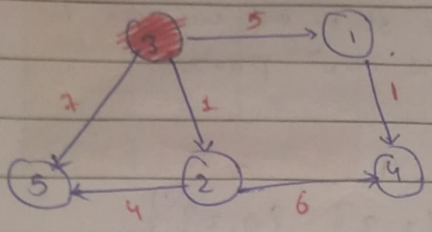
⇒ Code :-

```
pq.add(new pair(0,0));
while(pq.size()>0) {
    pair top = pq.remove();
    for (pair p : adj.get(node)) {
        totaldist = top.dist + p.dist;
        if (dist[p.node] > totaldist){
            dist[p.node] = totaldist;
            pq.add(new pair(p.node, total));
        }
    }
}
```

⑧ LeetCode Q. No. 743 { Network Delay time.

Page No.: YOUVA
Date:

Ex ①



K = 3

→ kth tower is transmitting Signal. Find the minimum time in which the signal can transmit to All towers.

at t = 0 → 3

t = 1 → 2, 3

t = 2 → 2, 3

t = 3 → 2, 3

t = 4 → 2, 3

t = 5 → 3, 2, 5, 1

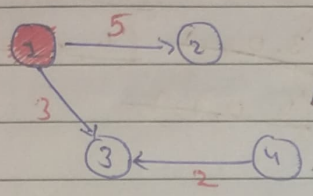
t = 6 → 3, 2, 5, 1, 4

1	2	3	4	5
5	1	0	6	5

Max

Ans => 6

Ex ②.



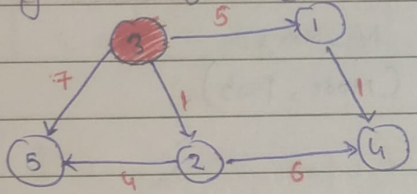
Ans = -1

Signals Are not able to reach to 4 tower so -1.

=> Dry Run:- edges = {{3,1,5}, {3,2,1}, {3,5,7}, {2,5,4}, {2,4,6}, {1,4,1}}

Time

Edges



Adj :-

Node

time

0 :-

1 : (4,1)

2 : (5,4), (4,6)

3 : (1,5), (2,1), (5,7)

4 :

5 :

0	1	2	3	4	5
Ans = X	5 ∞	1 ∞	0	7 6 ∞	7 5 ∞

Ans => 6 → Max

(3,0) → (1,5) 5 ✓

→ (2,1) ~~7~~ ✓

→ (5,7) 7 ✓

(2,1) → (5,4) 5 ✓

→ (4,6) 7 ✓

(5,5) ✗

(4,6) ✗

(4,7) ✗

(5,7) ✗

(1,5) → (4,1) 6 ✓

~~(3,0)~~, ~~(1,5)~~, ~~(2,1)~~, ~~(5,7)~~

~~(5,5)~~, ~~(4,7)~~, ~~(4,6)~~.

minheap.

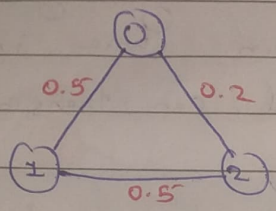
<node, time>

⑧ LeetCode Q.No. (1514) { Path With Maximum Probability.

0 =< p <= 1

st = 0, end = 2 } → from st to end we need to find max^m prob.

n = 3 → 0 to 3-1.



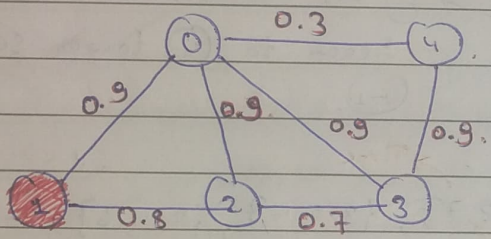
edges = { {0,1}, {1,2}, {0,2} }
(0) (1) (2)

succ = { 0.5, 0.5, 0.2 }.

Adj => 0 : (1, 0.5), (2, 0.2)
1 : (0, 0.5), (2, 0.5).
2 : (0, 0.2), (1, 0.5)

~~(1,1)~~ ~~(0,0.5)~~
~~(2,0.8)~~ ~~(2,0.81)~~
~~(3,0.81)~~ ~~(4,0.27)~~
~~(4,0.72)~~

=> Dry Run :- st => (1), end => (4).



Ans =

0	1	2	3	4
0.9 ~~Ø~~ Ø	1	~~0.8~~ 0.81 Ø	0.81 Ø	~~0.27~~ 0.72 Ø

MaxHeap
(Node, Prob).

(1,1) → (0, 0.9) 1*0.9 ⇒ 0.9 ✓
→ (2, 0.8) 1*0.8 ⇒ 0.8 ✓

(0, 0.9) → (1, 0.9) 0.81 ✗
→ (2, 0.9) 0.81 ✓
→ (3, 0.9) 0.81 ✓
→ (4, 0.3) 0.27 ✓

(2, 0.8) ✗

(3, 0.81) → (0, 0.9) 0.72 ✗
→ (2, 0.7) 0.56 ✗
→ (4, 0.9) 0.72 ✓

(2, 0.81) → (0, 0.9) 0.72 ✗
→ (1, 0.8) 0.64 ✗
→ (3, 0.7) 0.56 ✗

(4, 0.72) → (0, 0.3) 0.21 ✗
→ (3, 0.9) 0.63 ✗

(4, 0.27) ✗

Ans =

0	1	2	3	4 → end
0.9	1	0.81	0.81	0.72 → Ans

(10) LeetCode Q.No (1631) } Path with Minimum Effort.

Up, Left, Down, Right.

1	2	2
3	8	2
5	3	5

1 — 3 — 5 — 3 — 5

2 2 2 2 ⟹ max ⇒ (2) ⇒ effort.

1 → 3 — 8 — 2 — 2 — 2 → 5

2 5 6 0 0 3

max ⇒ effort.

1 — 2 — 2 — 2 — 5

1 0 0 3 ⟹ max ⇒ (3) ⇒ effort.

Arr

	0	1	2
0	1	2	2
1	3	8	2
2	5	3	5

	0	1	2
0	0	1 ∞	1 ∞
1	2 ∞	5 6 ∞	1 ∞
2	2 ∞	2 ∞	2 8 ∞

Ans

(0,0,0), (0,1,1)

(1,0,2), (1,1,6)

(0,2,1), (1,2,1)

(2,2,3), (1,2,3)

(2,0,2), (2,1,2)

(2,2,2)

MinHeap

{row, col, dist} ⇒ Triplet

(0,0) ↓0 → (0,1) (1) ⇒ 1 ✓ ; → (1,0) (2) ⇒ 2 ✓

(0,1) ↓1 → (0,0) (1) ⇒ 1 ✗ ; → (1,1) (6) ⇒ 6 ✓ ; → (0,2) (0) ⇒ 1 ✓

(1,2) ↓1 → (0,2) (0) ⇒ 1 ✗ ; → (1,1) (6) ⇒ 6 ✗ ; → (2,2) (3) ⇒ 3 ✓

(2,0) ↓2 → (1,0) (2) ⇒ 2 ✗ ; → (2,1) (2) ⇒ (2) ✓

(0,2) ↓1 → (0,1) (0) ⇒ 1 ✗ ; → (1,2) (0) ⇒ 1 ✓

(1,0) ↓2 → (0,0) (2) ⇒ (2) ✗ ; → (1,1) (5) ⇒ 5 ✓ ; → (2,0) (2) ⇒ 2 ✓

(2,1) ↓2 → (2,2) (2) ⇒ 2 ✓ ; → (1,1) (5) ⇒ 5 ✗ ; → (2,0) (2) ⇒ (2) ✗

(2,2) ↓2 ⟶ return this.

(r-1, n-1)

(r, c-1) ← (r, c) → (r, c+1)

(r-1, c)

(r+1, c)

Up Left Down Right.

r = { -1, 0, 1, 0 }

c = { 0, -1, 0, +1 }

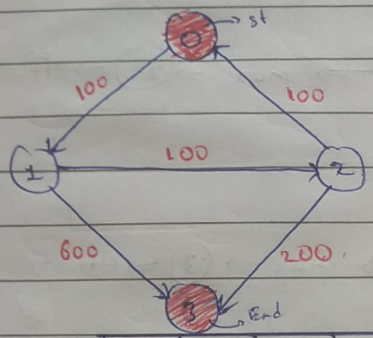
(11) LeetCode Q.No. (787) { Cheapest Flights Within K Stops

M T W T F S S | Page No.: | Date: | YOUVA

Ex (1):

K+1 Stops ⟶ Including dst.

Src = 0, dst = 3, k = 1



Ans =

0	100	200	700
0	1	2	3

(0,0,0) → (1,100) (100) ✓ ; (2,200) (200) ✓ ; (3,700) (700) ✓

(1,100,1) → (3,600) (700) ✓ ; (2,100) (200) ✓

(2,200,2) → (3,200) (400) ✗

(3,700,2) → Break

dst ; stops = k+1

MinHeap (Node, Cost, Stops)

(0,0,0), (1,100,1), (3,700,2), (2,200,2), (3,400,3) ✗

stops => k+1 → So Add the Nahi hoga

⇒ This will Not work ① Observation :-

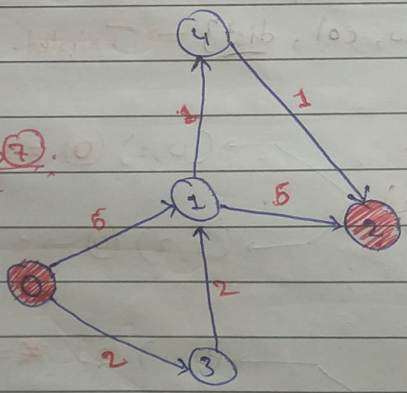
Sorting based on Cost will Give wrong Ans. b/c it will consider min cost. We need consider Stops Also

* Create minHeap based on stops.

Ex (2) :-

n = 5 flights {{0,1,5}, {1,2,5}, {3,1,2}, {0,3,2}, {1,4,1}, {4,2,7}}

Src = 0, dst = 2, k = 2



Ans => (7) should be

0	1	2	3	4
0	4	9	2	5

(0,0,0), (1,5,1), (3,2,1), (1,4,2), (2,9,3), (4,5,3)

MinHeap (Node, Cost, Stops)

(0,0,0) → (1,5) (5) ✓ ; (3,2) (2) ✓

(1,4,2) → (2,5) (9) ✓ ; (4,1) (5) ✓

(3,2,1) → (1,2) (4) ✓

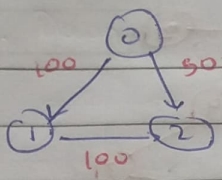
(1,5,1) → (2,5) (10) ✗ ; (4,1) (6) ✗ → Mistake Algo will Not Conside this It Need to be

(4,5,3) ✗ (k+1)

(2,9,3) (dist) → Break. → Return → But Wrong Ans. Connect to (7)

(2)

⇒ Observations:-



(0,0,0) → (1, 100) (100) ✓
(0,0,0) → (2, 500) (500) ✓

(1, 100, 1) → (2, 100) (200)

(0,0,0), (1, 100, 1)
(2, 500, 1), (2, 200, 2)

MinHeap
(Node, Cost, Stops)

Src ⇒ 0, dst = 2, k = 1.

Ans =

0	1	2
0	100 ~~∞~~	~~500~~ 200 ~~∞~~

(2, 500, 1) → return

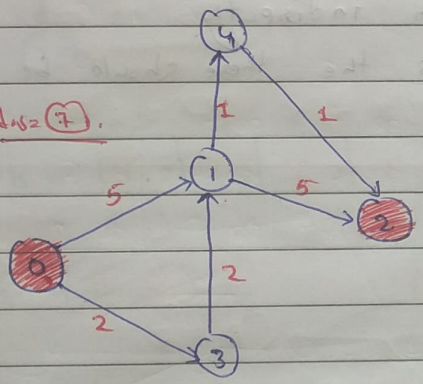
dst.

⇒ This is wrong Don't return this return Ans[dst] or break the Loop.

⇒ (3) Observation.

n = 5 flights = {{0,1,5}, {1,2,5}, {0,3,2}, {3,1,2}, {1,4,1}, {4,2,1}}

Src = 0, dst = 2, k = 2



Ans = (7).

(0,0,0) → (1,5) (5) ✓
(0,0,0) → (3,2) (2) ✓

(1,5,1) → (2,5) (10) ✓
(1,5,1) → (4,1) (6) ✓

(3,2,1) → (1,2) (4) ✓

0	1	2	3	4
0	~~5~~ 4 ~~∞~~	~~10~~ 7 ~~∞~~	2 ~~∞~~	6 ~~∞~~

~~(4,5,3)~~
~~(2,7,3)~~
~~(1,4,2)~~
~~(4,6,2)~~
~~(2,10,2)~~
~~(3,2,1)~~
~~(1,5,1)~~
~~(0,0,0)~~

(2, 10, 2) → ✗

(4, 6, 2) → (2, 1) (7) ✓

(4, 5, (3))

(1, 4, 2) → (4, 1) (5) ✓
(1, 4, 2) → (2, 5) (9) ✗

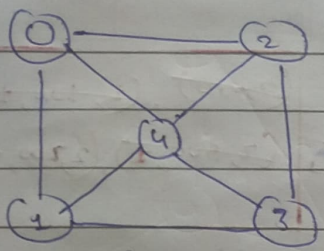
(2, 7, (3))

~~k~~ 3 = k+1 → continue

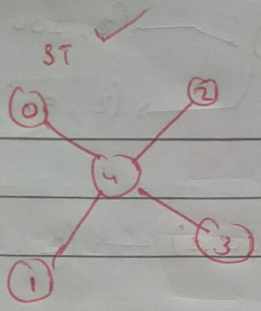
Ans[dist] ⇒ (7) // → Correct

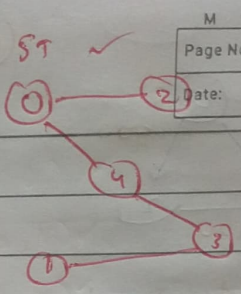
Obs:- First ~~triplet~~ entered in ~~queue~~ minHeap is Going ~~the one which entered~~ to be removed first, so First in First Out (Fifo), so instead of PQ we Normal Queue Not Need of Custom Comparable

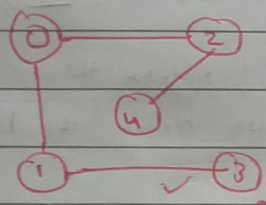
Spanning Tree :-



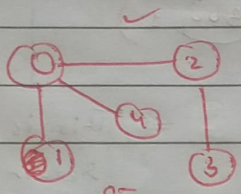
Graph.



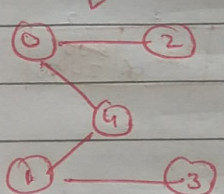




ST.



ST



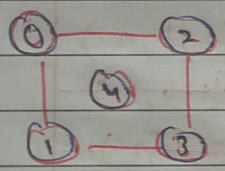
ST

⇒ Spanning Tree ⟶ There Should be One Component.

⟶ Every Node should be connected to each other directly or indirectly.

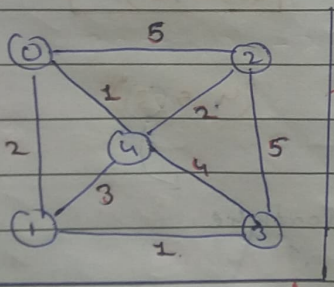
⟶ if there are 'n' Nodes the there should be 'n-1' Edges.

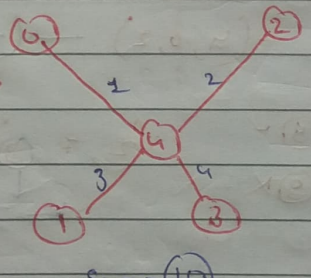
Undirected Graph.



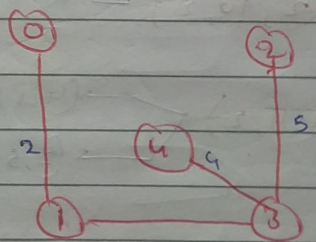
Not An Spanning Tree.

Minimum Spanning Tree :- ⟶ Undirected weighted Graph with One Connected Component

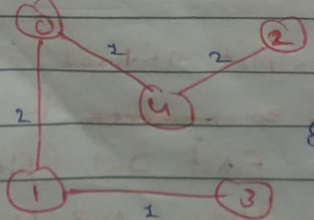




Sum = (10)



Sum = (12)



Sum = (6)

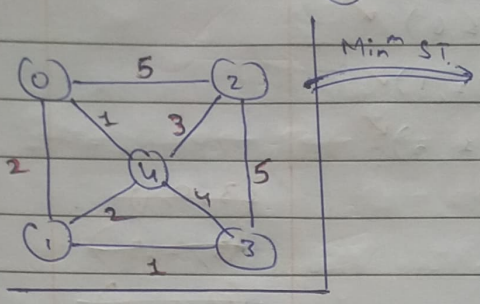
Minimum

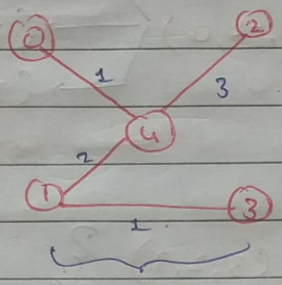
⟶ This is Minimum Spanning Tree.

Sum of edges weight is minimum

Prim's Algorithm :-

=> Minimum Spanning Tree





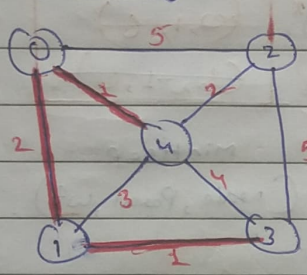
0 → (4,1)
1 → (3,1), (4,2)
2 → (4,3)
3 → (1,1)
4 → (1,2), (2,3).

edges = { (0,4,1), (2,4,3), (1,4,2), (1,3,1) }

Or

'Min^m ST.

=> Prim's Algo. :- → Dry Run ①.



Sum => ~~0 1 3 4~~ (6)

F → Insert kardo.

(0,-1,0)
- (1,0,2) ✓
- (4,0,1) ✓
- (2,0,5) ✓

(4,0,1)
- (1F,4,3) ✓
- (2F,4,2) ✓
- (3F,4,4) ✓

(1,0,2)
- (3F,1,1) ✓
- (4T F,1,3) ✗

	0	1	2	3	4
Vis =	T F	T F	T F	T F	T F

~~MST = {(1,0,1)}~~

MST { (4,0,1), (1,0,2), (3,1,1), (2,4,2) }

Sum → 1 + 2 + 1 + 2 => (6)

(3,1,1)
- (2F,3,5) ✓
- (4T,3,4) ✗

(2,4,2)
- (0T,2,5) ✗
- (3T,2,5) ✗

(1,4,3) ✗ → Already "T" Mark hai kuch Nahi Hoga ← (3,4,4) ✗

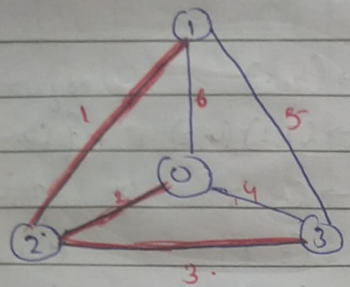
(2,0,5) ✗ (2,3,5) ✗

~~(2,3,5)~~
~~(3,1,1)~~
~~(3,4,4)~~
~~(2,4,2)~~
~~(1,4,3)~~
~~(4,0,1)~~
~~(2,0,5)~~
~~(1,0,2)~~
~~(0,-1,0)~~

MinHeap

(Node, parent, wt)

⇒ Prim's Algo → Dry Run (2).



MST = {(2,1,1), (0,2,2), (3,2,3)}

Vis = | 0 T | 1 T | 2 T | 3 T |

(1,-1,0) → (0, F, 1, 6) ✓ ; (2, F, 1, 1) ✓ ; (3, F, 1, 5) ✓

(2,1,1) → (0, F, 2, 2) ✓ ; (3, F, 2, 3) ✓

(0,2,2) → (1, T, 0, 6) ✗ ; (3, F, 0, 4) ✓

(3,2,3) → (0, T, 3, 4) ✗ ; (1, T, 3, 5) ✗

(3,0,4) ✗ (3,1,5) ✗ (0,2,6) ✗

~~(3,0,4)~~ ~~(3,2,3)~~ ~~(0,2,2)~~ ~~(3,1,5)~~ ~~(2,1,1)~~ ~~(0,1,6)~~ ~~(1,-1,0)~~

MinHeap (Node, Parent, wt)

---

(12) LeetCode Q. No. (1584) { Min Cost to Connect All Points

(x_1, y_1) | (x_2, y_2)

$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

$md = |x_1 - x_2| + |y_1 - y_2|$

(A,2,0) → (D, A, 13) ; (B, A, 4) ; (C, A, 7) ; (E, A, 7)

(3,A,4) → (C, B, 3) ; (D, B, 9) ; (E, B, 7)

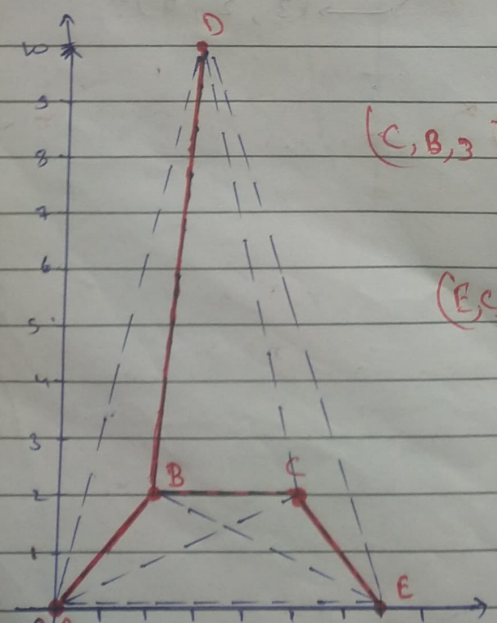
(C,B,3) → (A, C) ✗ ; (D, C, 10) ✓ ; (E, C, 4) ✓

(E,C,4) → (D, E, 14) ✓ ; (A, E) ✗ ; (B, E) ✗

(D,B,9) → A ✗ ; C ✗ ; E ✗

Sum = 0 → A → 4 → 7 → 11 → 20 → Ans

~~(D,E,14)~~ ~~(E,C,4)~~ ~~(D,C,10)~~ ~~(C,B,3)~~ ~~(D,B,9)~~ ~~(E,B,7)~~ ~~(E,A,7)~~ ~~(C,A,7)~~ ~~(D,A,13)~~ ~~(B,A,4)~~ ~~(A,2,0)~~



Vis = | A T | B T | C T | D T | E T |

Node, Parent, Dist

Bellman - Ford Algorithm

T.C ⇒ O(V * E)

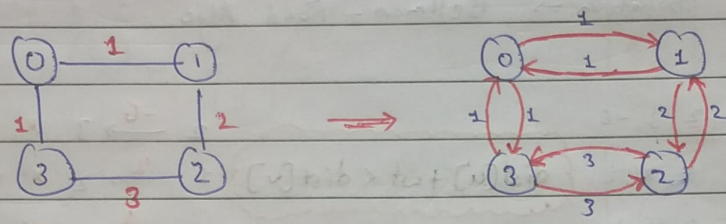
A.S ⇒ Depends on problem

↳ Shortest Distance from 'src' Node to Other Nodes in a dist Array

Dijkstra ⟶ directed, Undirected

Bellman ⟶ directed

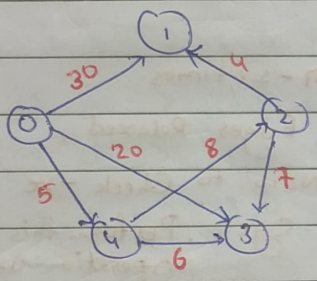
Negative Cycle ⟶ can be detected using Bellman Algo.



Shortest Distance {Edge - List} → n-1

src = 0

Edge ⟶ u, v, wt.

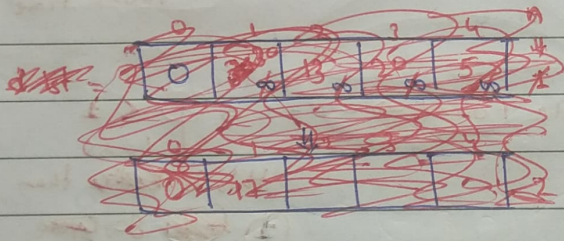


2 → 3 (7)
4 → 2 (8)
0 → 3 (20)
2 → 1 (4)
0 → 1 (30)
4 → 3 (6)
0 → 4 (5)

Need to perform all (n-1) times

[n = No. of Nodes]

2, 3, 7
4, 2, 8
0, 3, 20
2, 1, 4
0, 1, 30
4, 3, 6
0, 4, 5



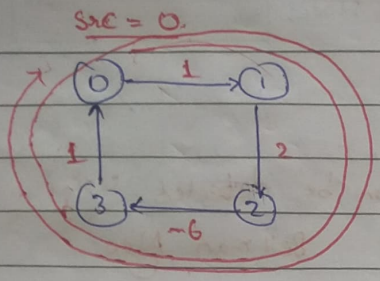
dist[u] + wt < dist[v]

	0	1	2	3	4	
dist =	0	30 ~~∞~~	∞	20 ~~∞~~	5 ~~∞~~	n=1
dist =	0	17 ~~30~~	13 ~~∞~~	11 ~~20~~	5	n=2
dist =	0	17	13	11	5	n=3
dist =	0	17	13	11	5	n=4

2 → 3 (7) ✗ ✗ ✗ ✗
4 → 2 (8) ✗ ✓ ✗ ✗
0 → 3 (20) ✓ ✗ ✗ ✗
2 → 1 (4) ✗ ✓ ✗ ✗
0 → 1 (30) ✓ ✗ ✗ ✗
4 → 3 (6) ✗ ✓ ✗ ✗
0 → 4 (5) ✓ ✗ ✗ ✗

LeetCode Q No. (743) {Network Delay Time} ⟶ Solved using this Algo.

1 ⇒ Negetive Cycle :-



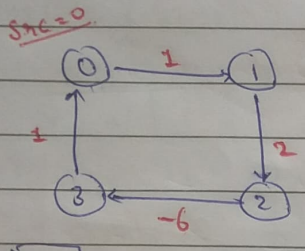
	0	1	2	3
dist =	~~∞~~ -4	-1 ~~∞~~	1 ~~∞~~	-5 ~~∞~~ -3

Negetive Cycle.

0 to 1 ⇒ 0 → 1 → 2 → 3 → 0 → 1 = -1

↳ 0 → 1 = 1.

Solution ⇒ Bellman – Ford Algo.



2, 3, -6

1, 2, 2 — dist[u] + wt < dist[v]

0, 1, 1

3, 0, 1

2 —(-6)→ 3 ✗ ✗ ✓ ✓

1 —(2)→ 2 ✗ ✓ ✓ ✓

0 —(1)→ 1 ✓ ✓ ✓ •

3 —(1)→ 0 ✗ ✗ ✓

n-1 Times

[n=4]

	0	1	2	3	
dist =	0	~~∞~~ 1	∞	∞	n=1.
dist =	0	1	~~∞~~ 3	∞	n=2.
dist =	~~∞~~ -2	1	3	~~∞~~ -3	n=3.
dist =	-2	1	3	-3	n=4

n-1 times Edges Relaxed, Now to Check -ve Cycle, Perform this operation One more time

↳ Negetive Cycle Found ⇒ -2 + 1 ⇒ -1 < 1

↓ Less

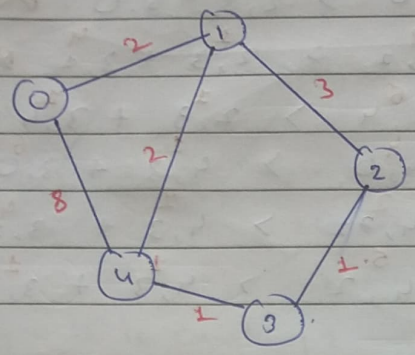
this is than

1 So, -ve Cycle found

Floyd-Warshall Algorithm.

↳ All Pairs Shortest Path.

'A' to 'B' throug 'C'



	0	1	2	3	4
0	0	2	5 ~~∞~~	5 ~~∞~~	4 ~~8~~
1	2	0	3	3 ~~∞~~	2
2	5 ~~∞~~	3	0	1	2 ~~∞~~
3	5 ~~∞~~	3 ~~∞~~	1	0	1
4	4 ~~8~~	2	2 ~~∞~~	1	0

Adj.

Go from A to B via/through C.

A, B = min((A, B), (A, C + C, B))

[No Need to check if i==j or i==C or j==C]

⇒ Pass (1) ⇒ C ⇒ 0

⇒ 1,2 > 1,0 + 0,2 ✗

⇒ 1,3 > 1,0 + 0,3 ✗

⇒ 1,4 > 1,0 + 0,4 ✗

⇒ 3,0 > 3,0 + 0,0 ✗

⇒ 3,1 > 3,0 + 0,1 ✗

⇒ 3,2 > 3,0 + 0,2 ✗

⇒ 2,1 > 2,0 + 0,1 ✗

⇒ 2,3 ≥ 2,0 + 0,3 ✗

⇒ 2,4 ≥ 2,0 + 0,4 ✗

⇒ 4,1 > 4,0 + 0,1 ✗

⇒ 4,2 > 4,0 + 0,2 ✗

⇒ Pass (2) ⇒ C ⇒ 1

⇒ 0,2 > 0,1 + 1,2 ✓

⇒ 0,3 > 0,1 + 1,3 ✗

⇒ 0,4 > 0,1 + 1,4 ✓

⇒ 3,0 > 3,1 + 1,0 ✗

⇒ 3,2 > 3,1 + 1,2 ✗

⇒ 3,4 > 3,1 + 1,4 ✗

⇒ 2,0 > 2,1 + 1,0 ✓

⇒ 2,3 > 2,1 + 1,3 ✗

⇒ 2,4 > 2,1 + 1,4 ✓

⇒ 4,0 > 4,1 + 1,0 ✓

⇒ 4,2 > 4,1 + 1,2 ✓

⇒ 4,3 > 4,1 + 1,3 ✗

⇒ Pass (3) ⇒ C ⇒ 2

⇒ 0,1 > 0,2 + 2,1 ✗

⇒ 0,3 > 0,2 + 2,3 ✗ ✓

⇒ 0,4 > 0,2 + 2,4 ✗

⇒ 1,0 > 1,2 + 2,0 ✗

⇒ 1,3 > 1,2 + 2,3 ✓

⇒ 1,4 > 1,2 + 2,4 ✗

⇒ 3,0 > 3,2 + 2,0 ✓

⇒ 3,1 > 3,2 + 2,1 ✓

⇒ 3,4 > 3,2 + 3,4 ✗

⇒ 4,0 > 4,2 + 2,0 ✗

⇒ 4,1 > 4,2 + 2,1 ✗

⇒ 4,3 > 4,2 + 2,3 ✗

⇒ Pass (4) ⇒ C ⇒ 3 :-

⇒ 0,1 > 0,3 + 3,1 ✗
⇒ 0,2 > 0,3 + 3,2 ✗
⇒ 0,4 > 0,3 + 3,4 ✗
⇒ 1,0 > 1,3 + 3,0 ✗
⇒ 1,2 > 1,3 + 3,2 ✗
⇒ 1,4 > 1,3 + 3,4 ✗

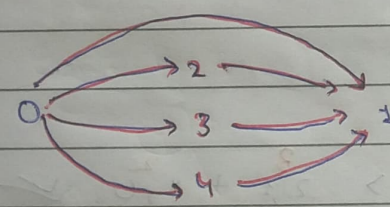
⇒ 2,0 > 2,3 + 3,0 ✗
⇒ 2,1 > 2,3 + 3,1 ✗
⇒ 2,4 > 2,3 + 3,4 ✓
⇒ 4,0 > 4,3 + 3,0 ✗
⇒ 4,1 > 4,3 + 3,1 ✗
⇒ 4,2 > 4,3 + 3,2 ✓

⇒ Pass (5) ⇒ C ⇒ 4.

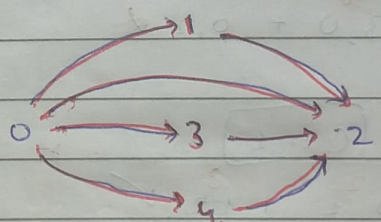
⇒ 0,1 > 0,4 + 4,1 ✗
⇒ 0,2 > 0,4 + 4,2 ✗
⇒ 0,3 > 0,4 + 4,3 ✓
⇒ 1,0 > 1,4 + 4,0 ✗
⇒ 1,2 > 1,4 + 4,2 ✗
⇒ 1,3 > 1,4 + 4,3 ✓

⇒ 2,0 > 2,4 + 4,0 ✗
⇒ 2,1 > 2,4 + 4,1 ✗
⇒ 2,3 > 2,4 + 4,3 ✗
⇒ 3,0 > 3,4 + 4,0 ✓
⇒ 3,1 > 3,4 + 4,1 ✓

0 to 2.



0 to 2.



0 → 3 → 2

0 → 3 + 3 → 2.

~~0 to 2~~

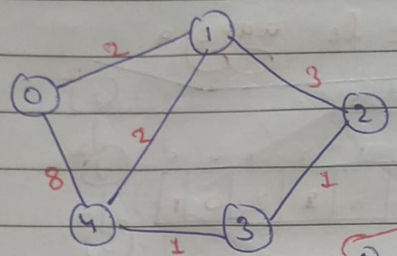
→ code

```
for(int k = 0 ; k < n ; k++){
    for(int i = 0 ; i < n ; i++){
        for(int j = 0 ; j < n ; j++){
            dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
        }
    }
}
```

* T.C ⇒ $O(n^3)$
* S.C ⇒ $O(n^2)$

(13) LeetCode Q. No (1334) { Find the City with Smallest No. of Neighbours At A Threshold distance

M T W T F S S | Page No.: | Date: | YOUVA



t = 2

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

Ans

0 → (1)

1 → (0), (4)

City which is connected to less No. of Cities with dist less than or equal to Threshold

2 → (3), (4)

3 → (2), (4)

4 → (1), (2), (3)

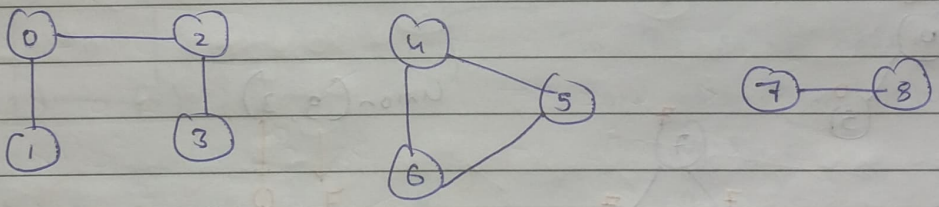
Disjoint Set Union (DSU) ⟶ Time Complexity.

↳ Undirected Graph ⟶ Cycle detect.

↳ No. of Connected Components

↳ Minimum Spanning tree

Adjacency Matrix, Adj List, Edge List.

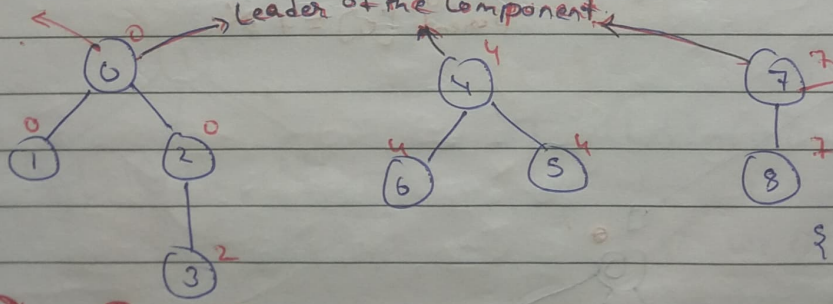


we will Not do this

edges = { {0,1}, {0,2}, {2,3}, {4,6}, {4,5}, {5,6}, {7,8} }

Group Leader

Leader of the Component



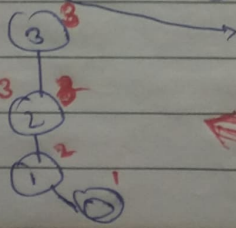
Group Leader

Group Leader same hai

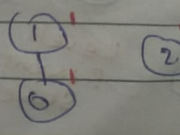
5 5

{4,6}, {5,4}, {6,5}

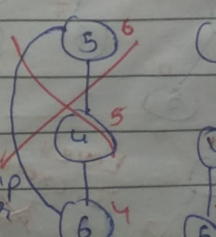
{0,1}, {0,2}, {2,3}



we will do union of '0' K Group leader or 2' K Group Leader Ka

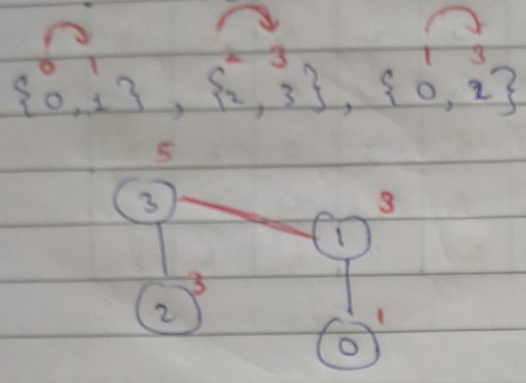


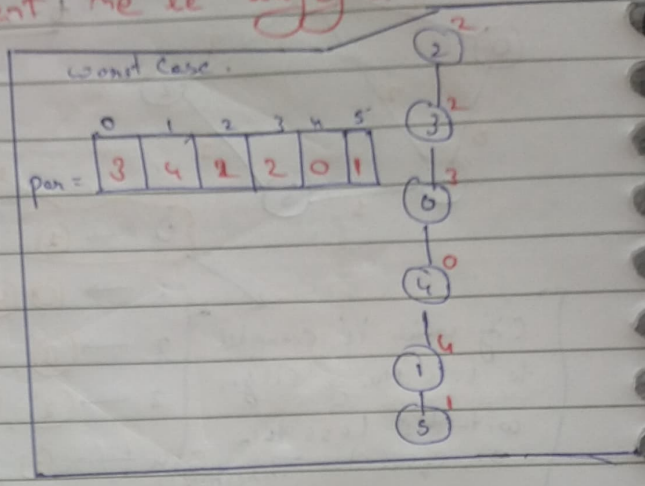
Union of this.



⇒ find(a) ⟶ Leader of 'a'

⇒ Union (a, b) ⟶ Ek Component : me le aayega.





⇒ Optimisation

Skewed Tree V/s Balanced Tree V/s. Two-Level Tree.

↓ ↓ ↓

Highest No. of Levels. ↓ find(a) ⇒ T.C ⇒ O(1)

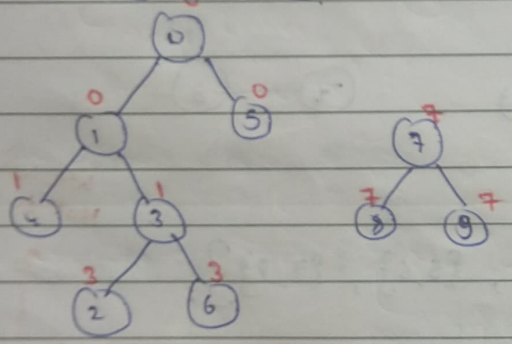
↓

find(a) → O(n) T.C

Less No. of Levels ↓

find(a) → O(log n)

⇒ Union By Size :-



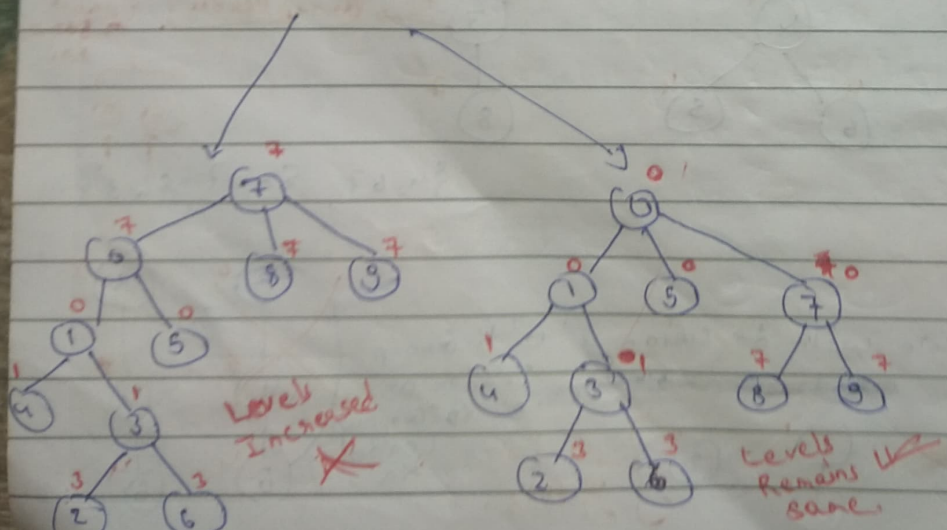
Union(9, 3)

7 0

parent[7] = 0 ✓

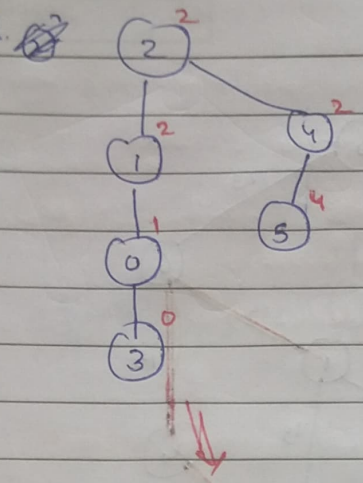
or

parent[0] = 7 ✗



Path Compression :-

Ex ①

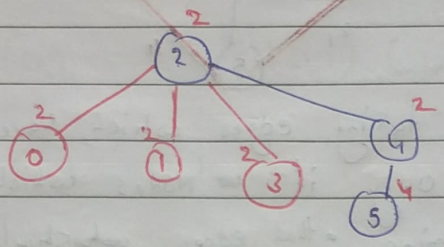


find(3).
find(0)
find(1)
find(2).

Leader → Recursion

Union(3,5)

During Back track Change the Connections



DSU Time Complexity

Adj List → O(E * Union) ≅ O(E).

O(α(n)) ~ O(4)

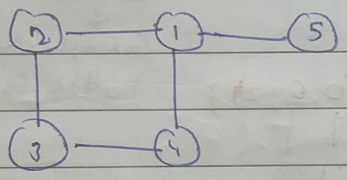
---

(14) LeetCode Q.No (684) { Redundant Connections }

n nodes

→ A Tree is that Graph where there are n-1 edges & No Cycles.

Ex: ①

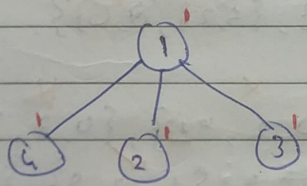


→ We Need to Convert this to Tree. To Do so we need to remove one edge which is part of Cycle.

edges = { {1,2}, {2,3}, {3,4}, {1,4}, {1,5} }

Cycle Detected

DSU

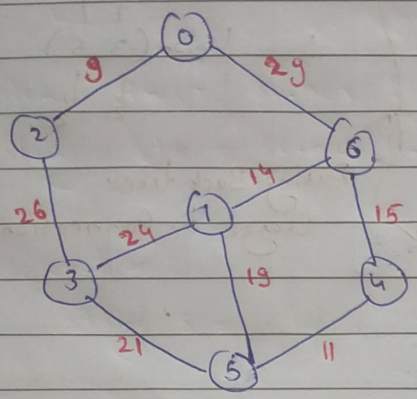


{1,4}

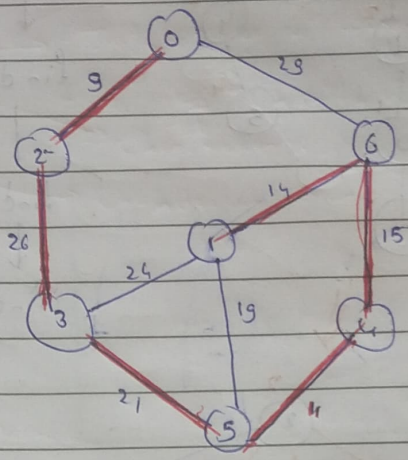
→ Leader(1) & Leader(4) are Already Same it means there is cycle. So we can remove this edge

Kruskal's Algorithm → Revise Prim's Algo

⇒ Minimum Spanning Tree.

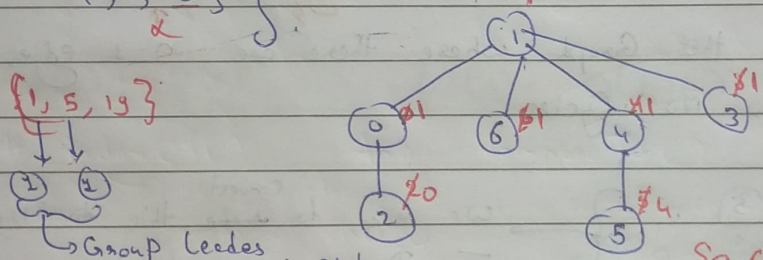


Original.



⇒ Start taking edges which have min^m wt. but only if Nodes which are forming that edges should have different different Group leader

{ {3,0,9}, {5,4,11}, {2,1,14}, {6,4,15}, {1,5,19}, {3,5,21}, {3,1,24}, {2,3,26}, {0,6,29} }



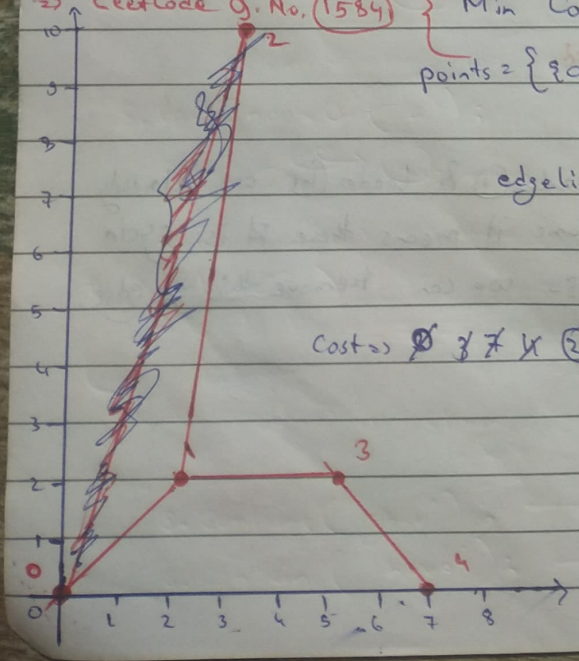
minCount ⇒ ~~1 2 3 4 5 6 7~~ 8

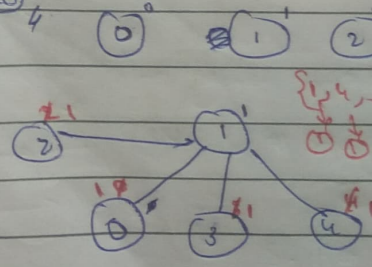
⇒ LeetCode Q. No. (1584) { Min Cost to Connect All points.

points = {{0,0}, {2,2}, {3,10}, {5,2}, {7,0}}

edgelist = {{0,1,4}, {0,2,13}, {0,3,7}, {0,4,7}, {1,2,9}, {1,3,3}, {1,4,9}, {2,3,10}, {2,4,14}, {3,4,4}}

Cost ⇒ ~~0 1 7 11~~ 20

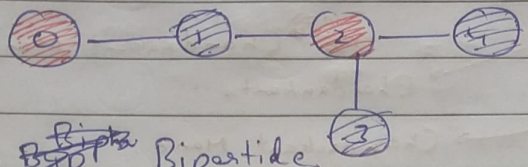




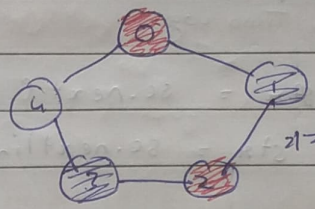
⇒ LeetCode Q. No (785) { Is Graph Bipartite. → Colouring of Graph

→ Non-Cyclic

→ Even Cyclic.

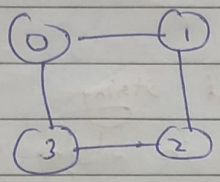


⇒ ~~Bipartite~~ Bipartide



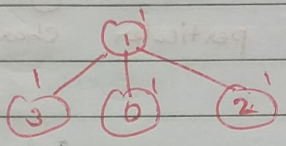
⇒ Not a Bipartite.

⇒ DSU ⟶ Can Only Detected Cycles ⟶ Not [Odd / Even]



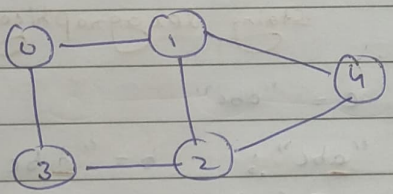
(0,1), (1,2), (2,3), (3,0)

Cycle Detected



⇒ DSU ⟹ parent, size, parity → true (red) / false (Blue)

Colour



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

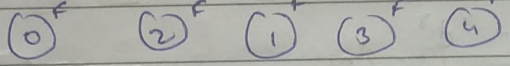
Not A Bipartite

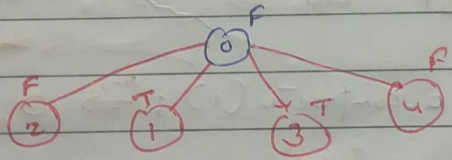
T / F

opp.

diff parity (Even)

Same Parity (Odd)



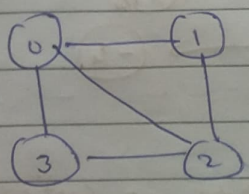


1, 4

T

Opp.

Depends on size of Component if (4) has more size Then It will be Opposite



0 → 3, 2, 1

1 → 0, 2

2 → 0, 3, 1

3 → 2, 0

→ Give Adj List & Check we Only want to Once for all ele

we will use This Condition to Check only once