VARDHMAN
PAGE NO.
DATE :

Graph :-

is a non-linear ds consisting of nodes & edges. nodes are also called vertex mainly graph consist of finite set of vertices (or nodes) & a set of edges that connect a pair of nodes.

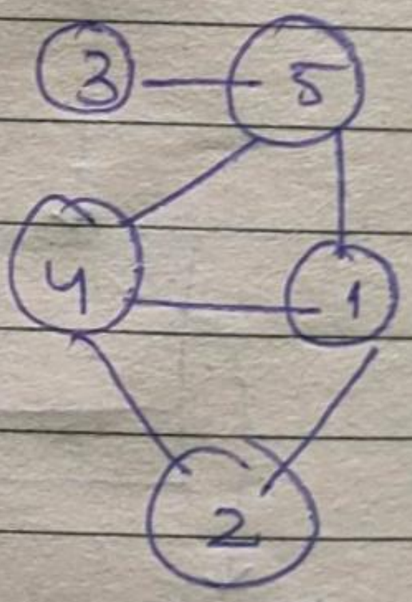
Representations :-

1. Adjacency Matrix
2. Adjacency List.

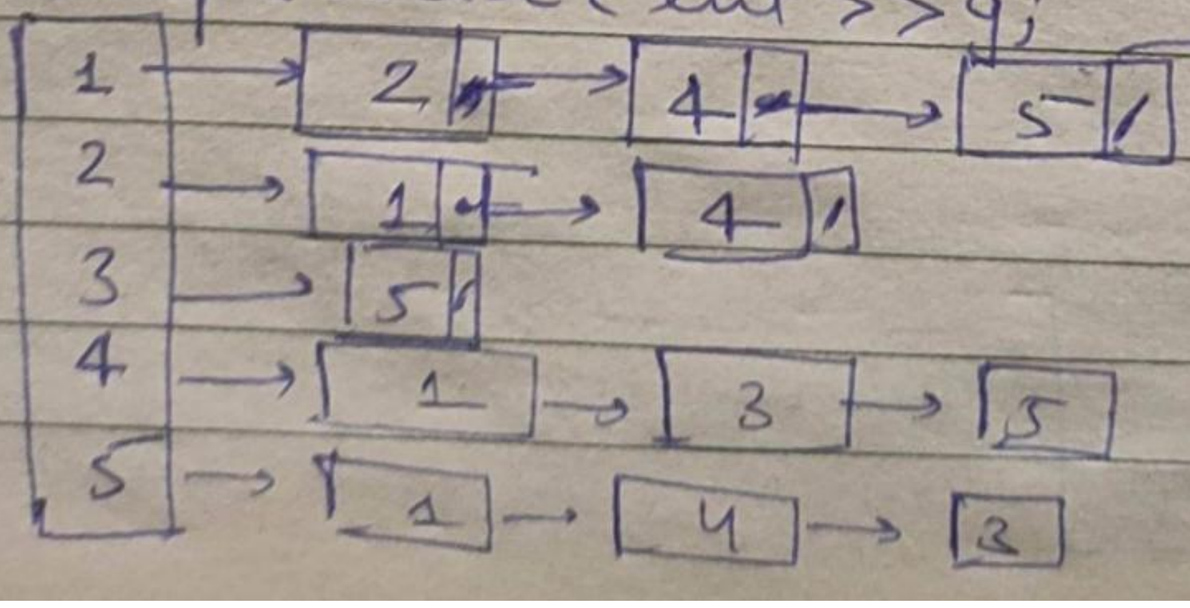
→ 2D Array of size V × V where V is the number of vertices. it also used for weighted graph. adj[i][j] = w, means edge from i to j with weight w.

	0	1	2	3
0	0	5	10	2
1	1	0	0	1
2	0	0	5	2
3	0	1	2	3

$O(V^2)$



Adjacency List: An array of list. size of arr = no: of vertices. we can also implement map<vector<int>> gj → LinkList

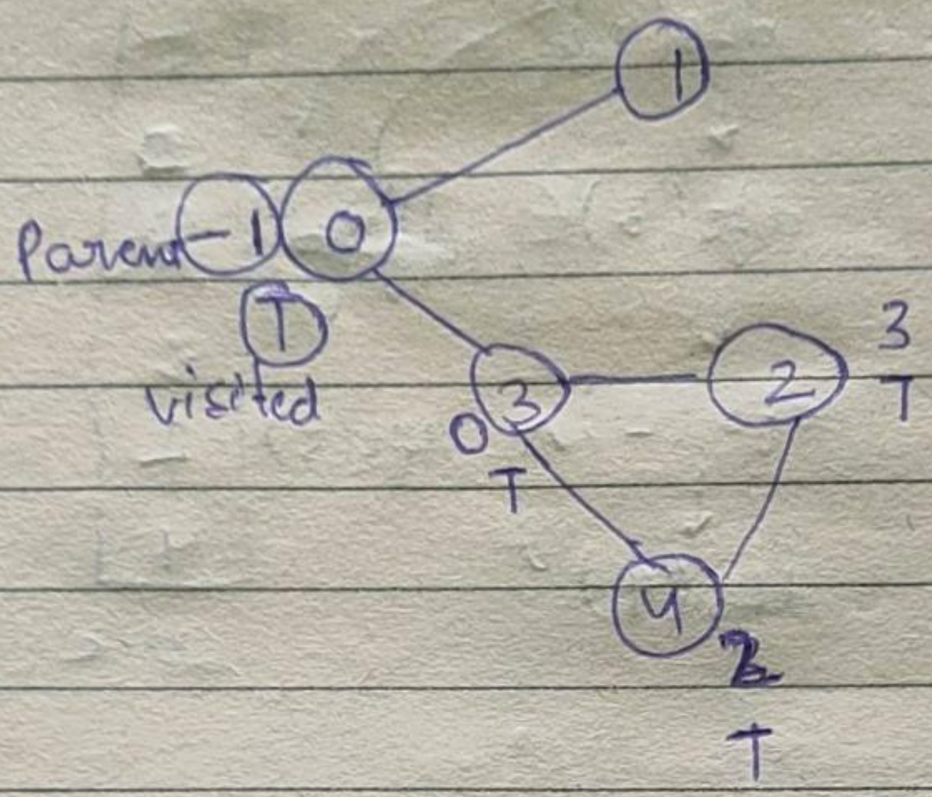


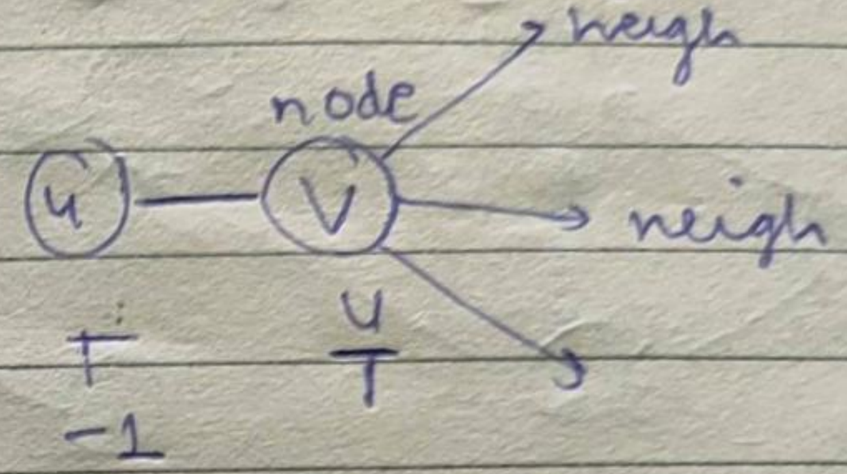
space
O(2E+V)
O(E+V)

VARDHMAN
PAGE NO.
DATE :

Cycle Detection :-

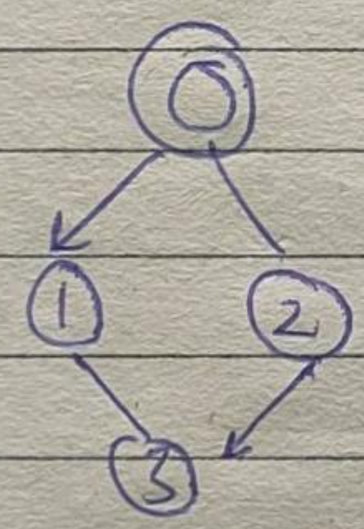
Undirected graph :- If we revist the same node hence cycle is there

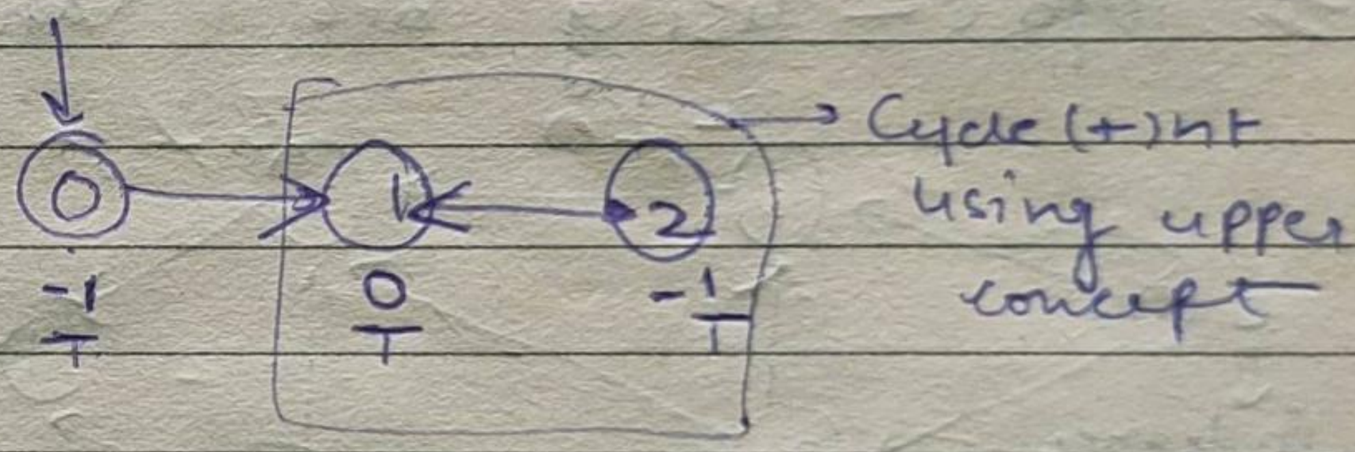




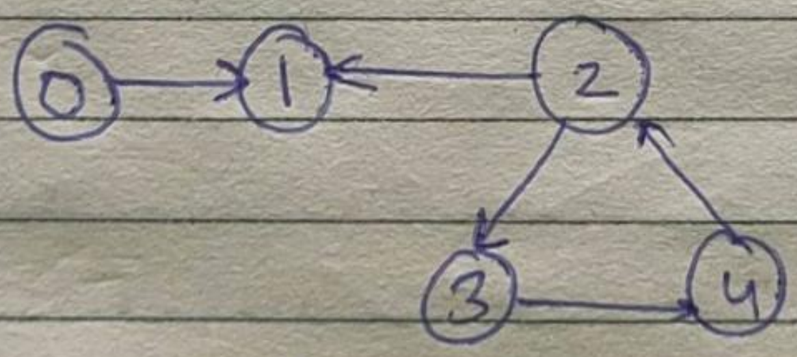
p[node] != neigh.

Directed Graph :-





P[2] != 1



call stack

dfs(0)

dfs(1)

dfs(2)

dfs(3)

dfs(4)

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

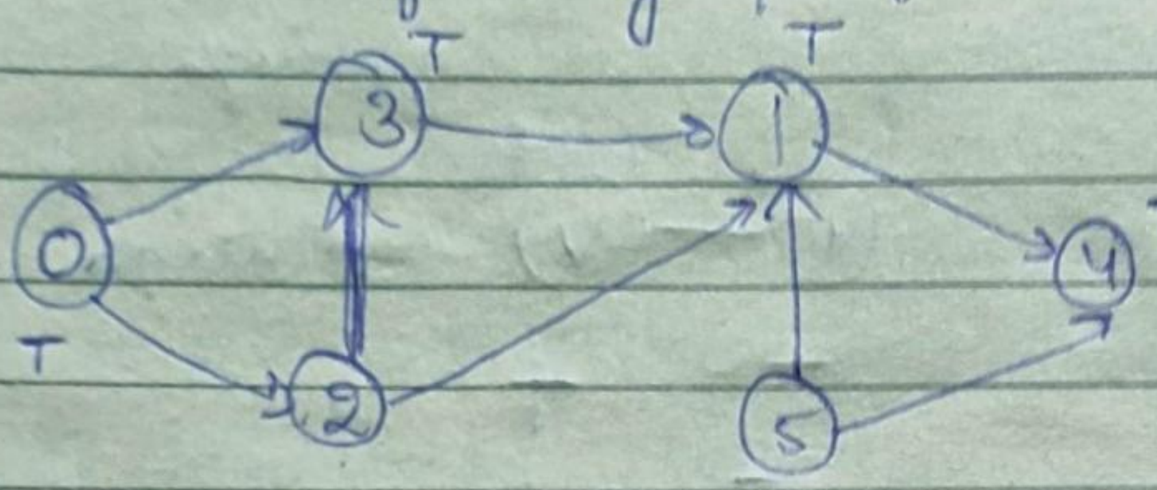
res[] = | 0 F | 1 F | 2 T | 3 T | 4 T |

recs[neight] = true.

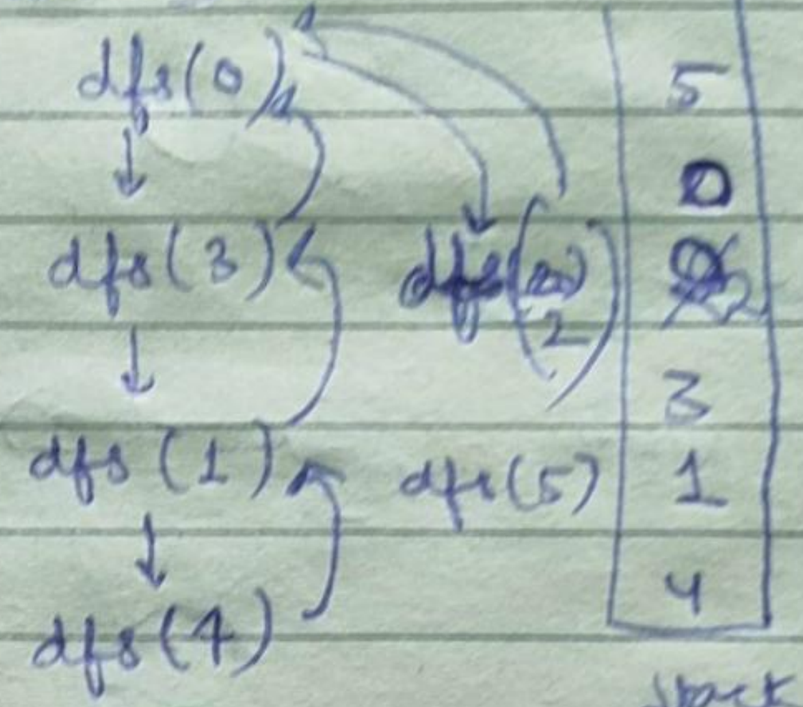
VARDHMAN
PAGE NO.
DATE :

Topological Sort :- write in linear form such that if any edge (u,v) u appears before v.

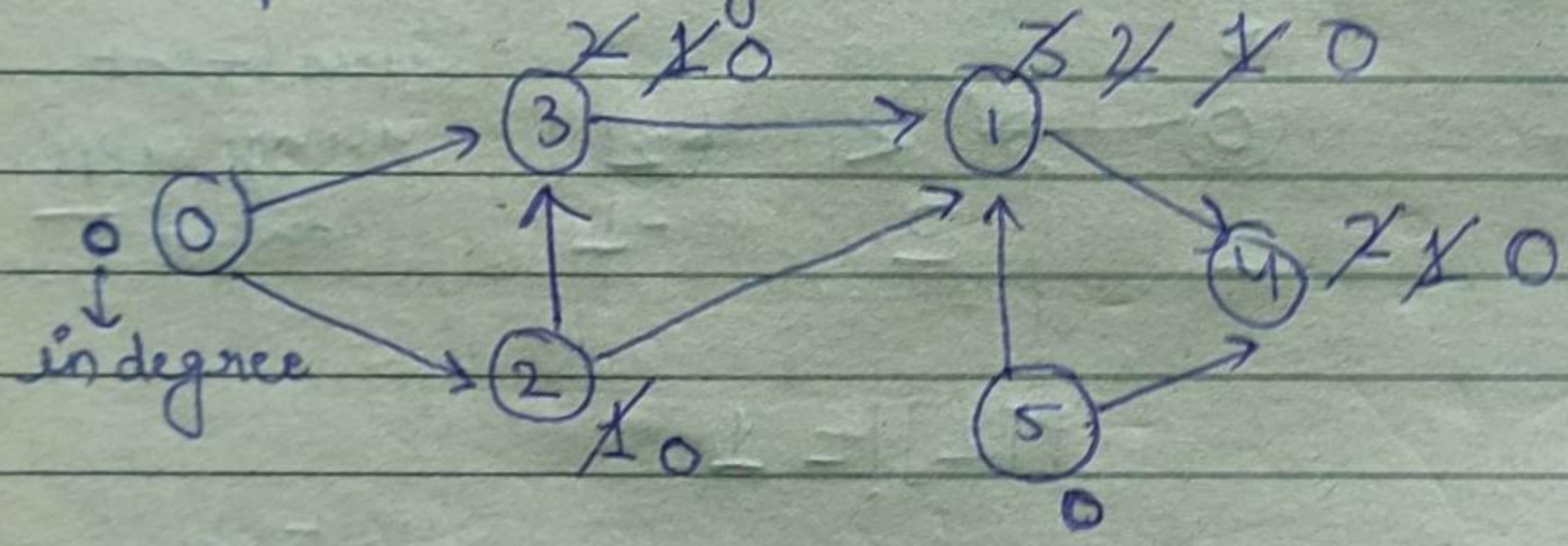
DAG
Directed Acyclic graph (means no revisit)



0 2 3 5 1 4
5 0 2 3 1 4 ✓



Topo s — using BFS (Kahn's algorithm)

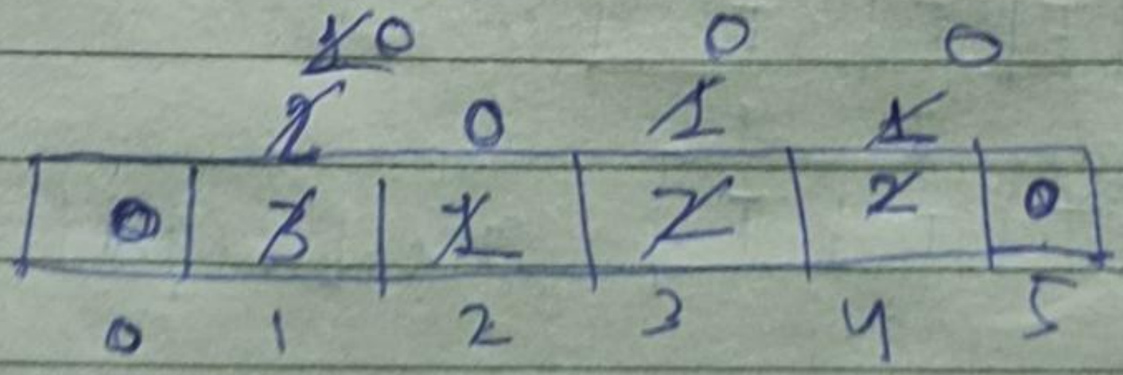


Queue

0	5	2	3	1	4

Ans

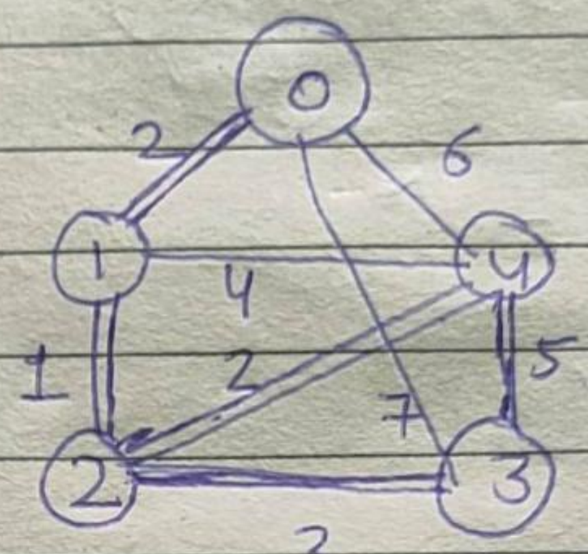
0	5	2	3	1	4



VARDHMAN
PAGE NO.
DATE :

Minimum Spanning Tree - visited
Prims Algorithm.

	0	1	2	3	4
vis (4)	T	T	T	T	T



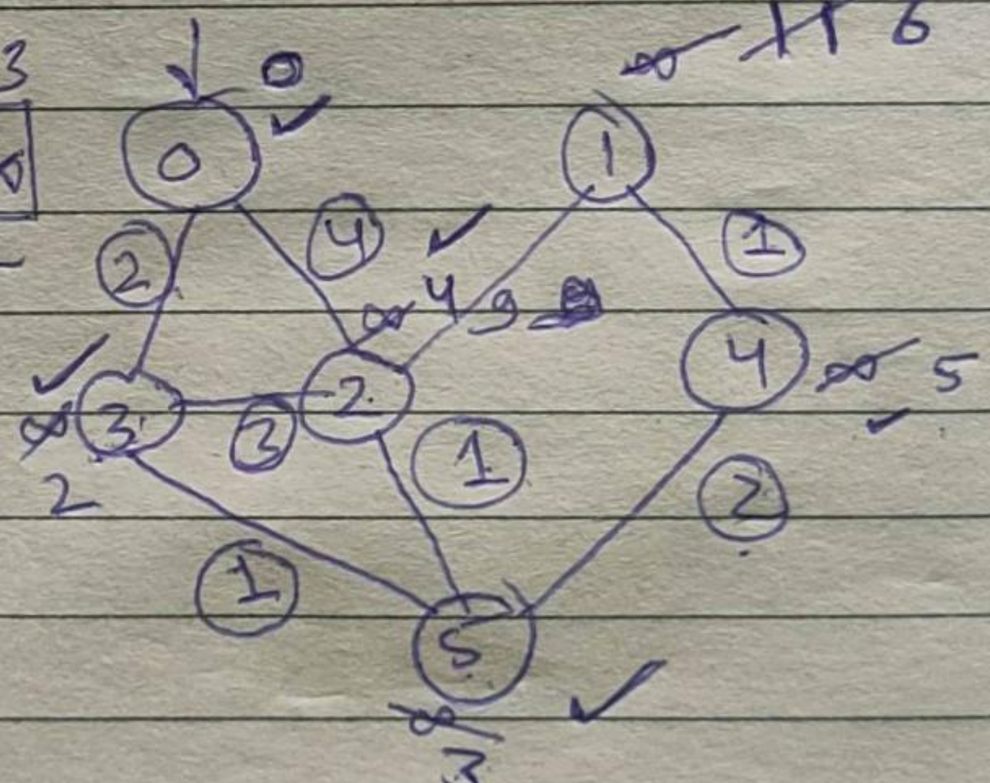
pq (0,0) <v, wt>
pq (1,2) (4,6)
(3,7)

extract min
pq (4,6) (3,7) (2,1) (4,4) ans → ~~8~~ 8
pq (4,6) (3,7) (4,4) (4,2) (3,3)
pq (4,6) (3,7) (4,4) (3,3) (3,5)

Use pq + hashmap for optimisation like (4,2) h pq m fir bhi (4,4) dalreh apan weight ki value modify kar sakte hn.

Dijkstra's algorithm :-

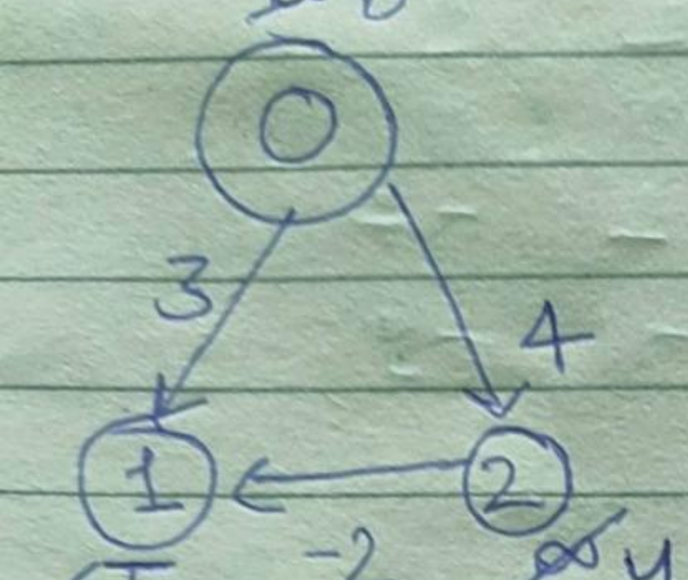
	0	1	2	3	4	5
ans	0	6	4	2	5	3



pq (0,0) (wt, v)
pq (2,3) (4,2)
pq (3,5) (4,2)
pq (4,2) (5,4)
pq (11,1) (5,4)
pq (11,1) (6,1) so if (visited[4] == true) continue
wt wt

pq (11,6) continue

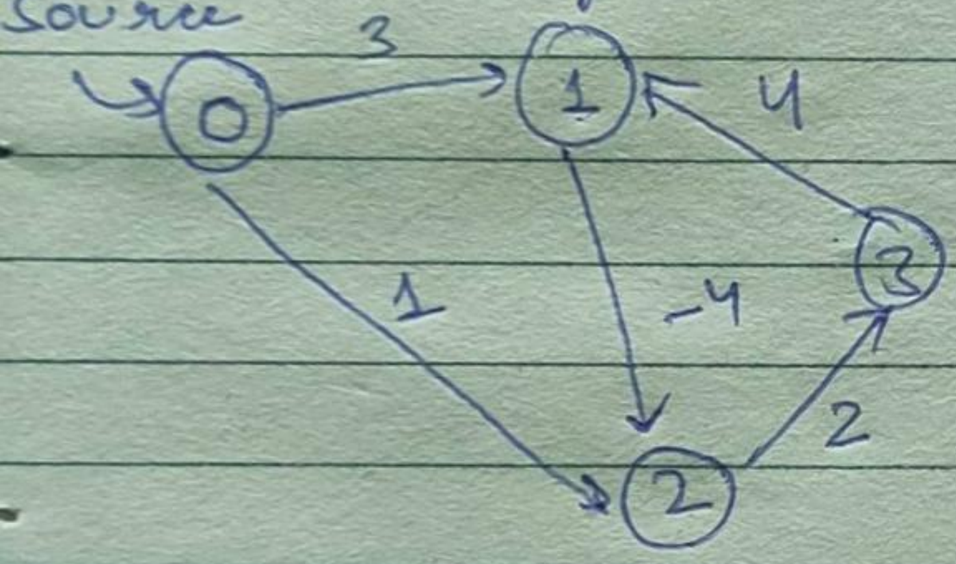
Bellman - ford algorithm



	Dijkstra	Bellman-ford
Negative edge	X	✓
	✗ fail	
Time complex	O(Elogv)	O(EV) (V-1)E

Dijkstra algo not works with negative edge.

Working



V →	0	1	2	3	
(V-1)					
count	0	∞	∞	∞	← dist
1	0	3	~~7~~ -1	1	
2		3	-1	1	
3		3	-1	1	

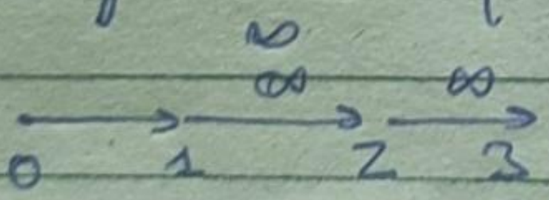
for every edge $(s \xrightarrow{wt} d)$

```
if (dist[s] + wt < dist[d]) {
    dist[d] = dist[s] + wt
}
```

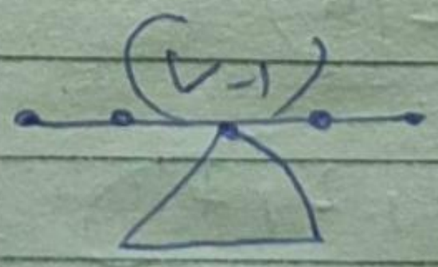
order of edge

0 → 1
0 → 2
1 → 2
3 → 1
2 → 3

Why it require V-1 iterations :-



2 → 3
1 → 2
0 → 1

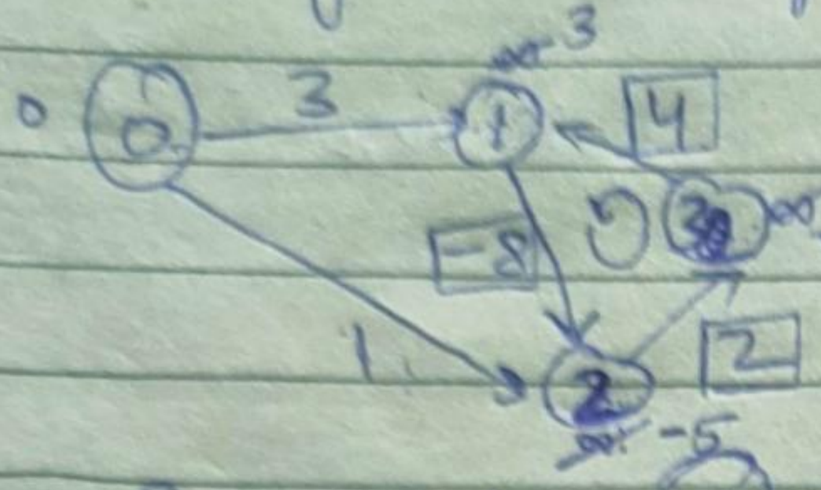


max longest edge dist (V-1) edge.

0 → 1 1 → 2
0 → 2 2 → 3
3 → 1

VARDHMAN
PAGE NO.
DATE :

Negative edge cycle :-



-8 + 4 + 2 = (-2) (-ve)

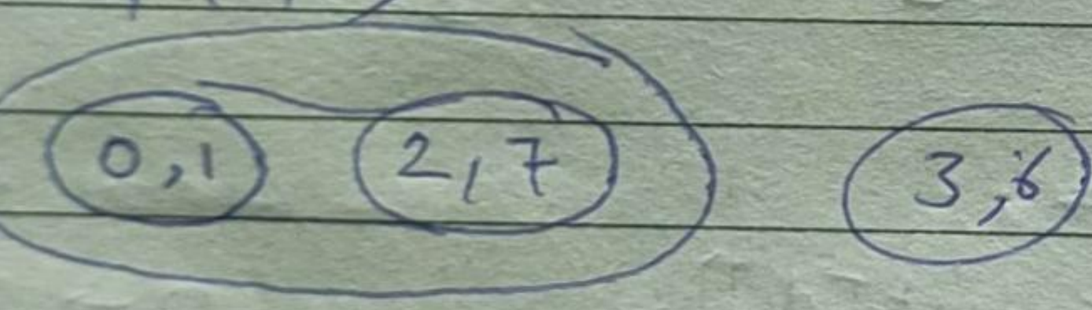
V →	0	1	2	3
.	0	∞	∞	∞
1	0	3	1	-7
2		-3	-11	-9
(V-1) 3				
↳ 4 changes.				

Disjoint sets :-

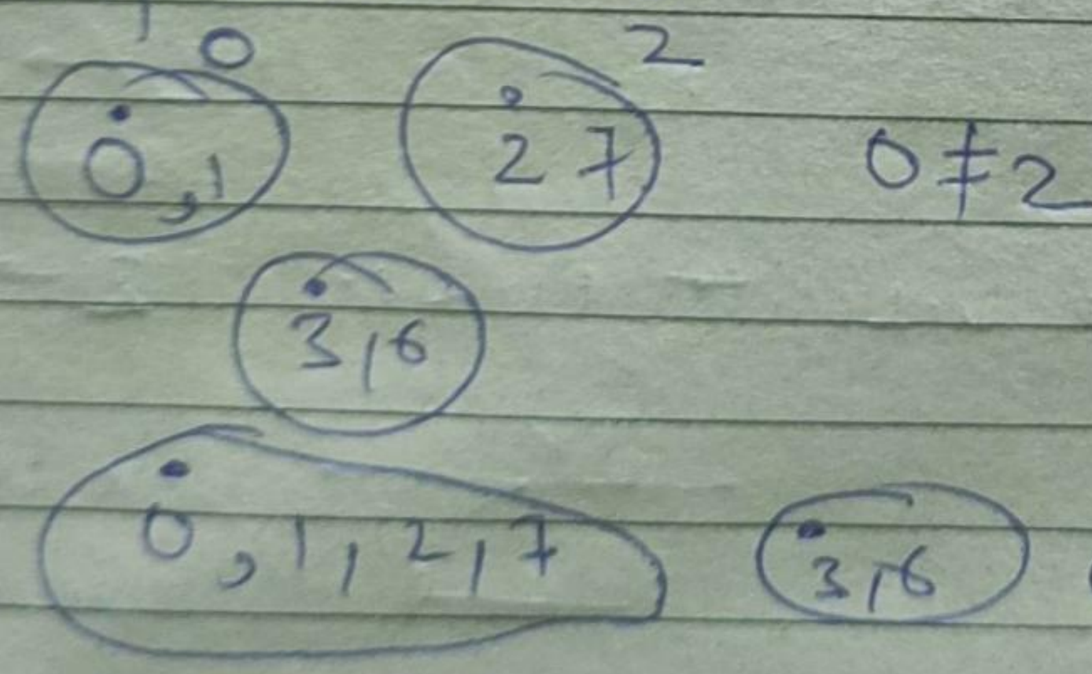
People : 0 1 7
4 3 2
5 6

operation
→ makefriend (a, b)
→ areFriend (a, b)

mf(0,1)
mf(2,7)
mf(3,6)
af(0,2) → F
mf(0,7)
af(0,2)

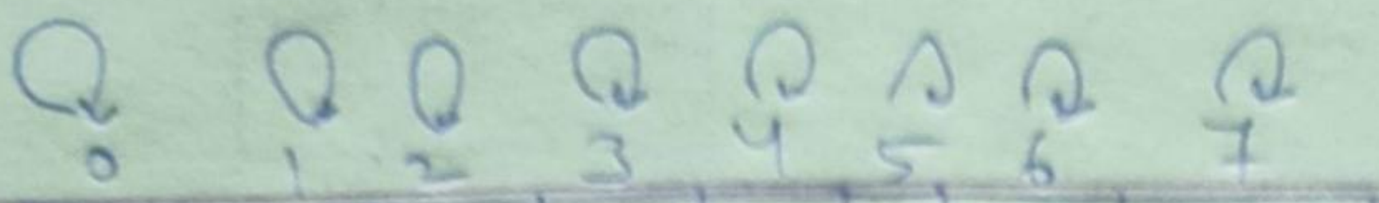


make representative



0 ≠ 2 F

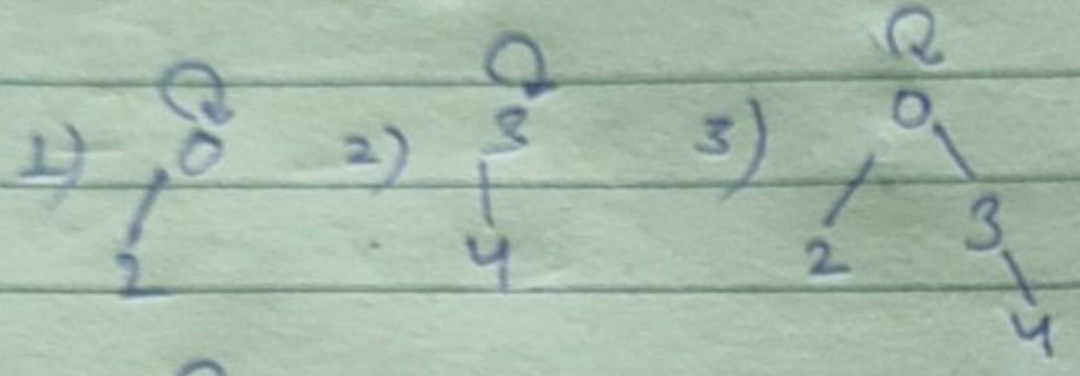
0 = 0 ⇒ T.



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

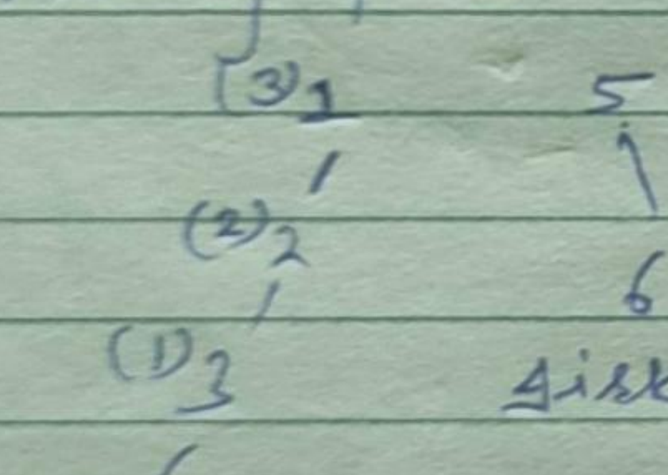
3 union(2,4) 4) union(5,4)

2→0 4→3 x→5 0



④ complexity depends of tree height.

so, make rank use while making parent



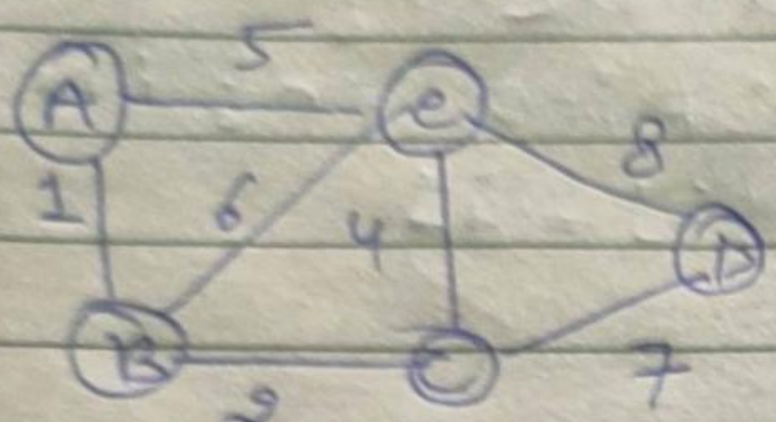
Jiski rank Badi hai vo parent

#/ Kruskal ako

```
int p[] = new int[n]
initialize() {
for (int i=0; i<n; i++){
    p[i] = i;
}
}
int find (int x) {
if (p[x] == x) return x;
return find(p[x]);
}
void union (int a, int b)
int x = find(a);
int y = find(b);
if (x == y) return;
p[y] = x;
}
```

VARDHMAN
PAGE NO.
DATE :

MST (Kruskal algo) connected UG



(V-1) edge

- A → B 1
- B → C 2
- E → C 4
- C → D 7

sorted edge	weight
AB	1
BC	2
EC	4
✗ AE	5
✗ BE	6
CD	7
✗ ED	8