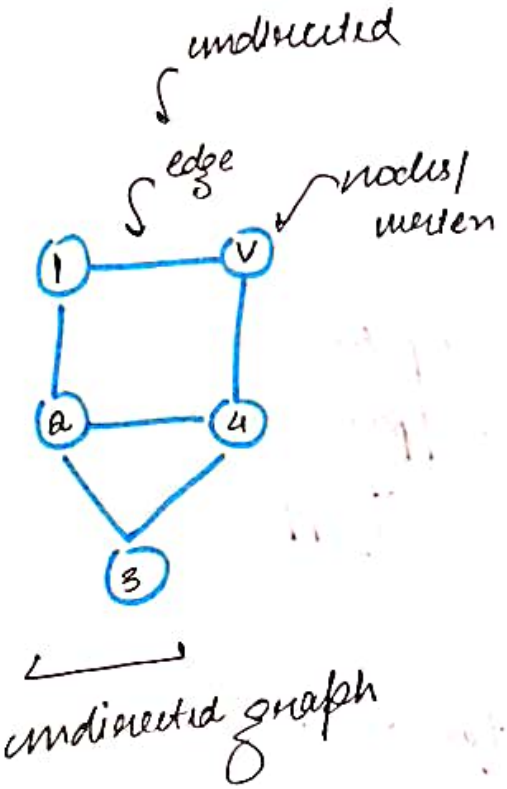
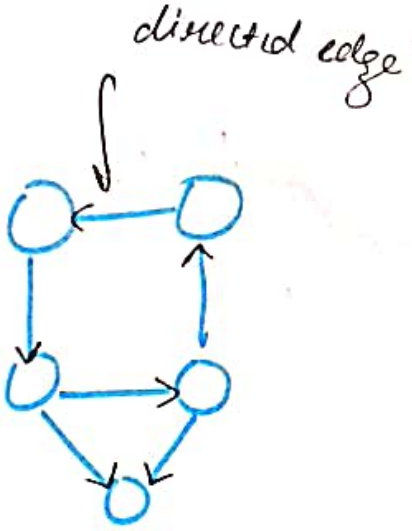
INTRODUCTION TO GRAPH



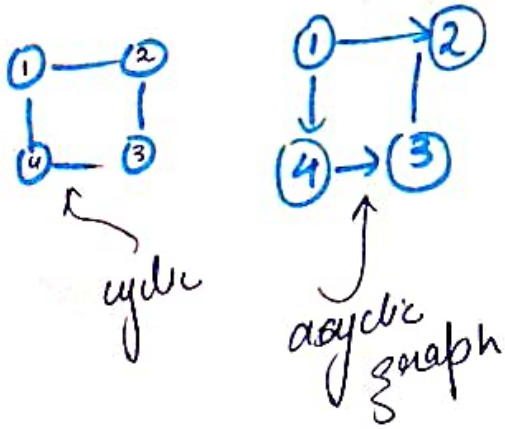
undirected graph



directed graph

CYCLES IN A GRAPH

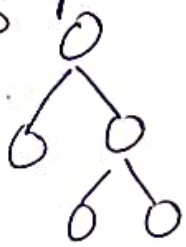
.. Start from node 1 end at that node..



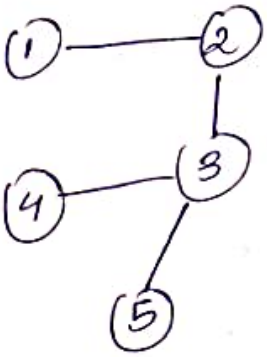
cyclic

acyclic graph

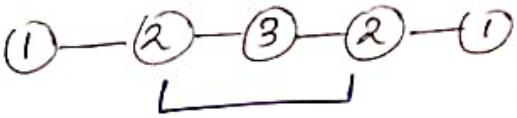
Binary tree also graph



Path: contains nodes & edges and each of them is reachable



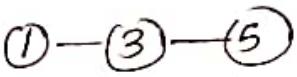
↑ path



↖ X path

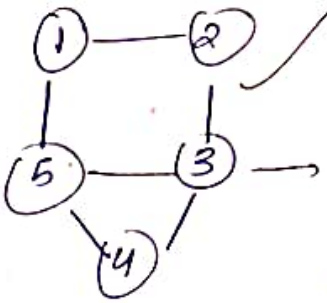
2 appear twice

* a node cannot appear twice



↳ X path not per direct relation in (1) and (3)

Degree:



no of edges attached

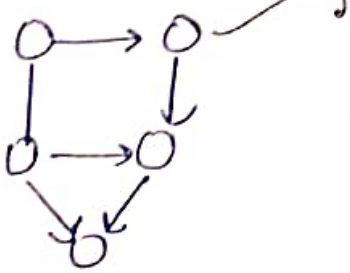
O(3) = 3

total degree of graph = 2x no of edges

as every edge associates with two nodes.

* when undirected

* Diwali

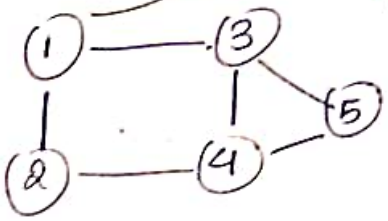


indegree ( ) = 1
out degree = 1

GRAPH REPRESENTATION

input → n nodes
m edges
5, 6

MATRIX WAY → adjacency matrix

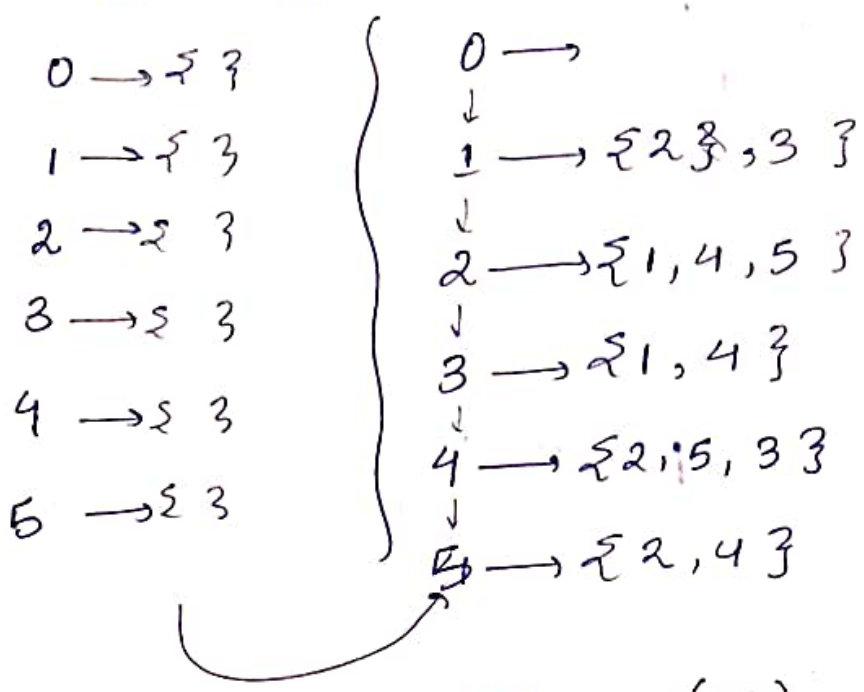


one based :. adj[n+1][n+1]

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	1	1	0	
2	—	1			1	
3		1			1	1
4			1	1	✱	1
5				1	1	

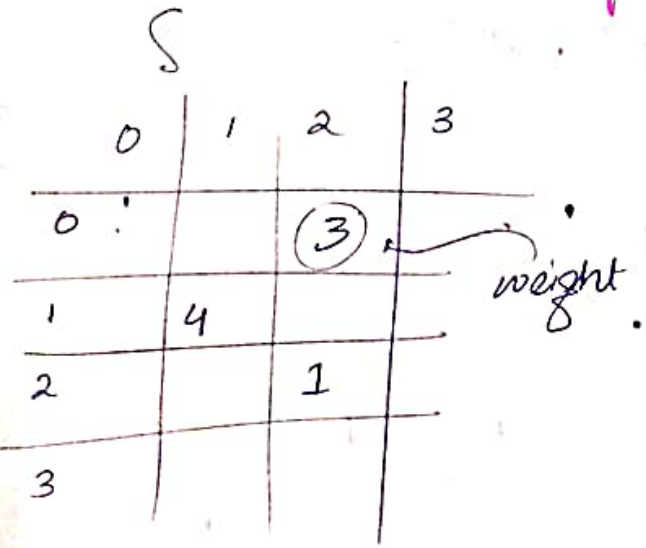
Space → nxn

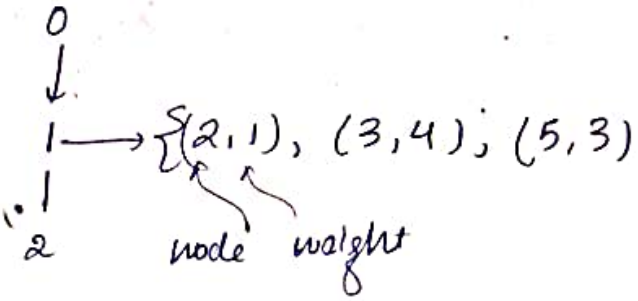
LIST WAY → adj[n+1]



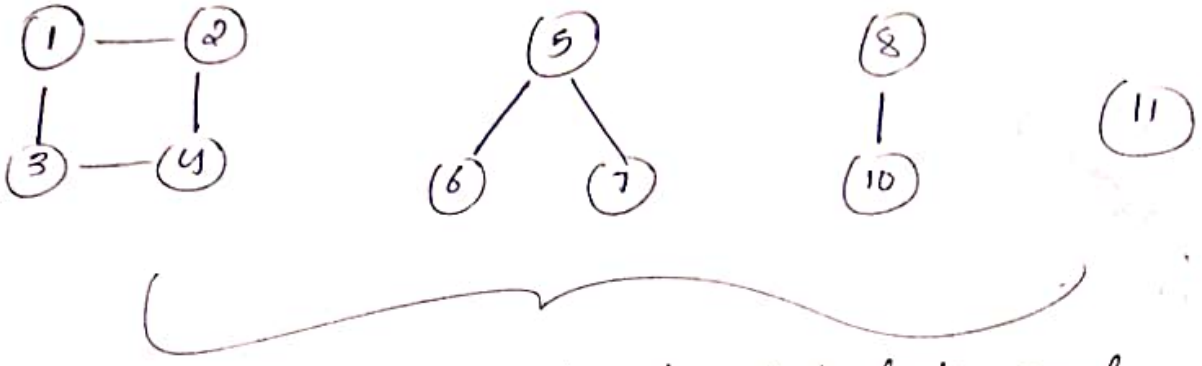
SPACE - O(2E)

WEIGHTED GRAPH





CONNECTED COMPONENTS



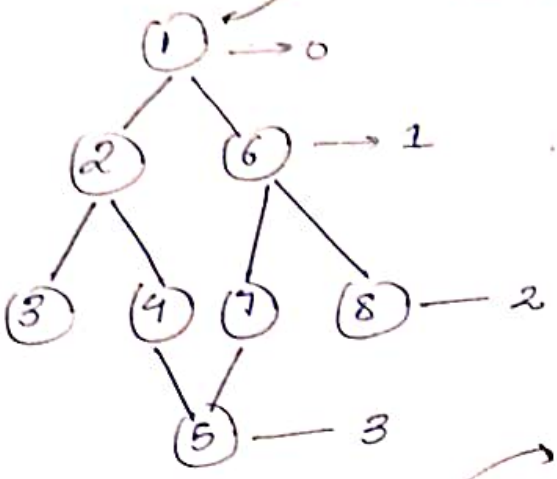
can be treated like single graph

and these individual components are connected components of graph

BFS TRAVERSAL

level wise

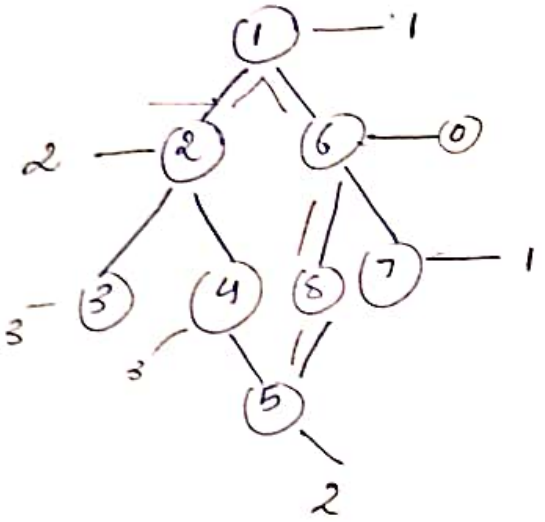
1-based



1 2 6 | 3 4 7 8 | 5

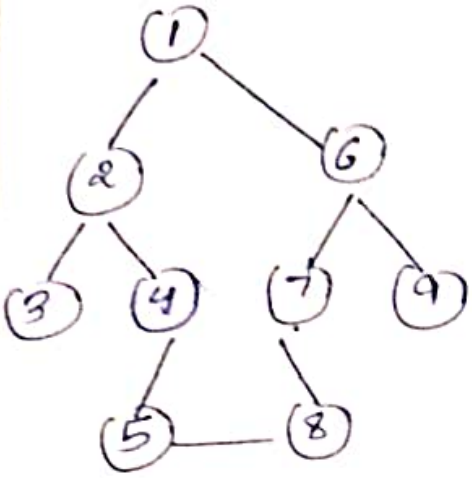
0 1 2 3

1 starting point



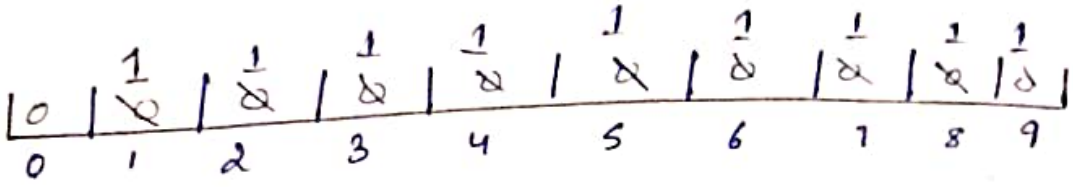
6 | 1 7 8 | 2 5 | 3 4

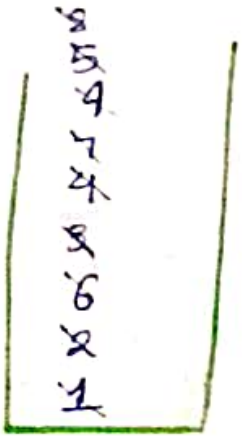
0 1 2 3



starting node = 1

visited





Queue

visit node

mark as visited

then visit all non unvisited neighbour of that node

1 2 6 3 4 7 9 5 8

vis array

arraylist bfs

queue

q.add(0)

vis(0) = true;

O(n)

while (!q.isEmpty()){

Integer node = q.poll();

bfs.add(node);

for (Integer it : adj.get(node)){

if(!vis[it]){

q.add(it);

vis[it] = true

}

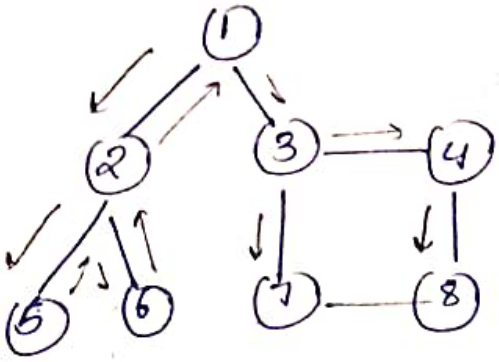
}

total dequee times (for every node)

Time → O(n) + O(2E)

Space →

...DEPTH FIRST SEARCH...



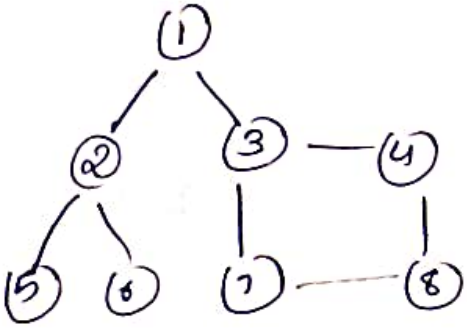
Starting = 1

go any neighbour node

then visit its neighbour node.

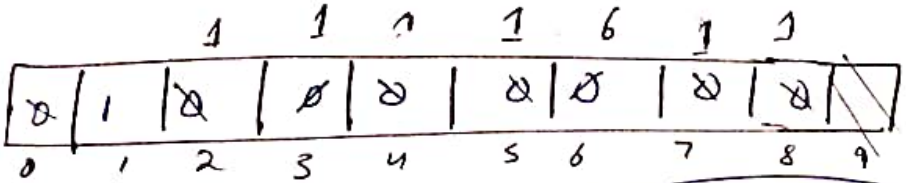
1 2 5 6 3 7 8

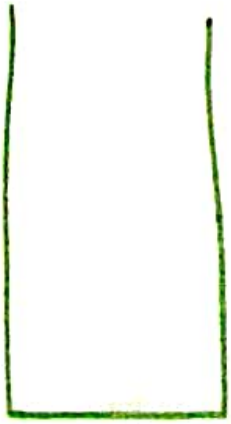
Starting = 3



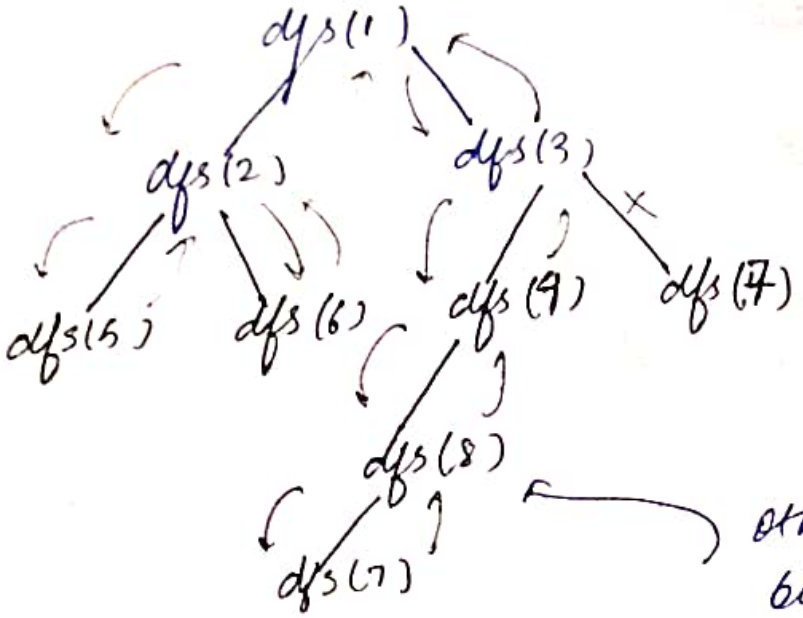
3 → 4 - 8 - 7 - 1 - 2 - 5 - 6

vis





```
dfs(node) {
    vis[node] = 1
    list.add(node)
    for(int it : adj(node)) {
        if unvisited → dfs(it)
    }
}
```



others nodes also but visited is not shown

outer

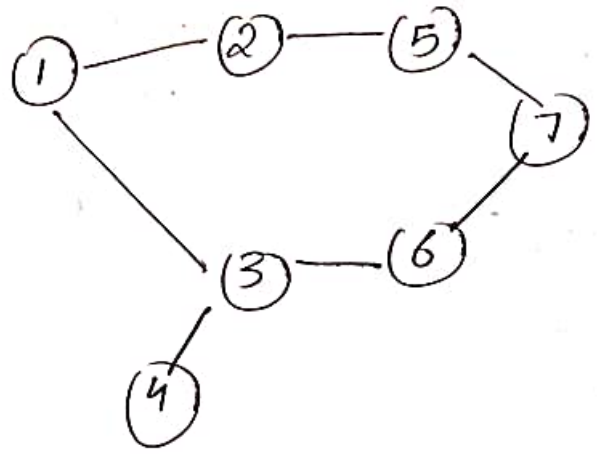
Time - $O(n) + O(\text{total degree} = 2E)$

Space - $O(n)$

inner loops runs for 2E times

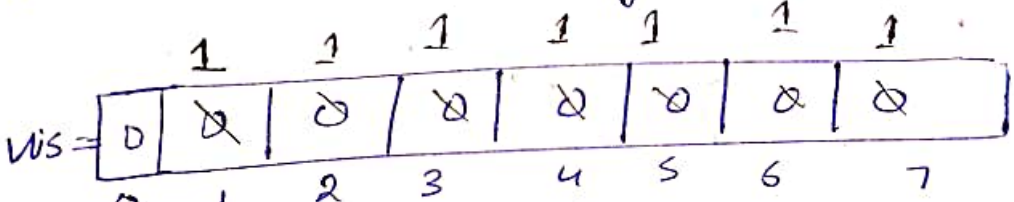
1, 2, 5

CYCLE DETECTION



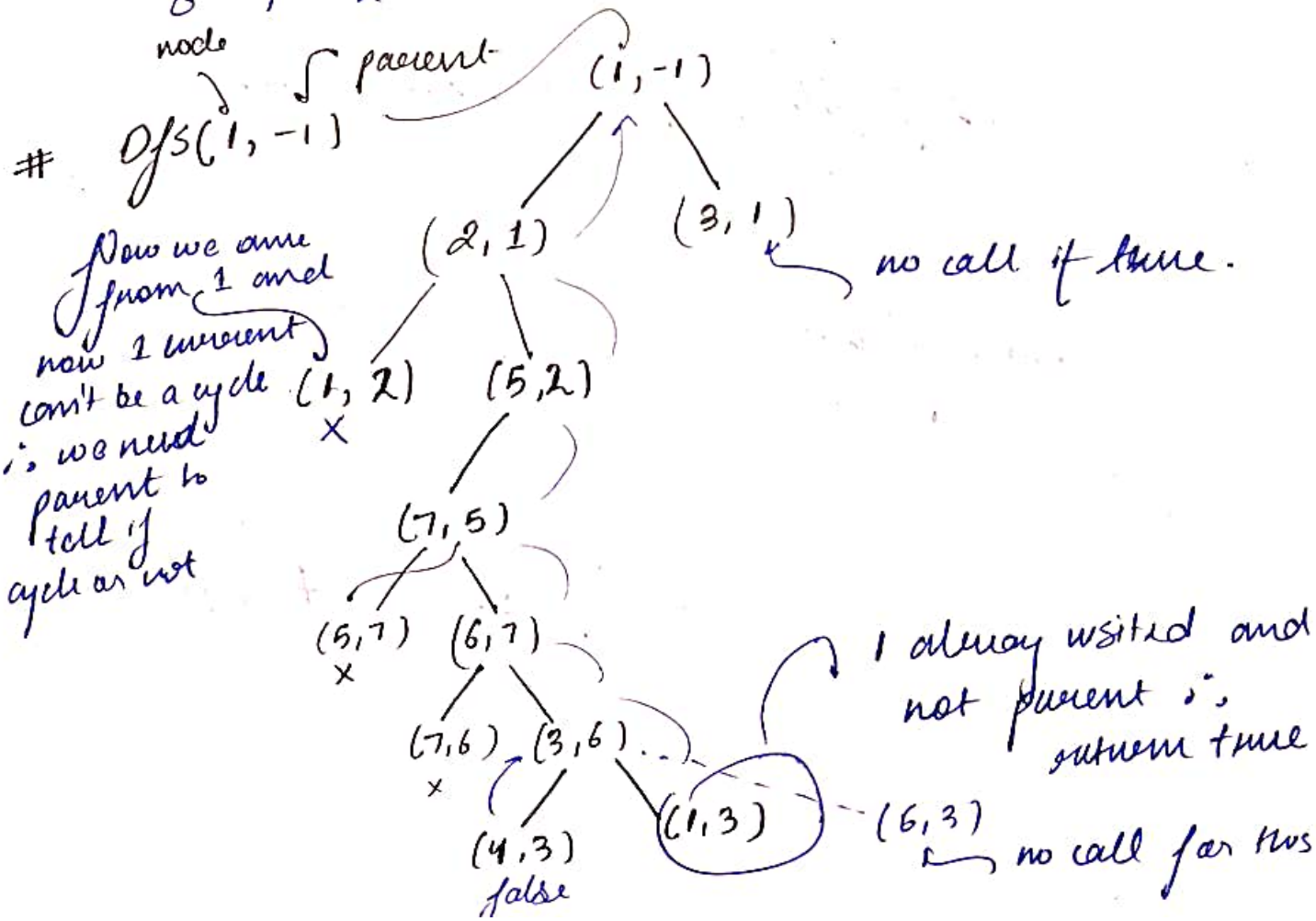
1 → {2, 3}
2 → {1, 5}
3 → {4, 1, 6}
4 → {3}
5 → {2, 7}
6 → {3, 7}
7 → {5, 6}

Start from somewhere if we get previously visited node then → A cycle is there.



node, parent

dfs(1, -1)



do dfs if any not which is not parent node and visited also then cycle detected.

→ for (i = 1 → nodes) { when more than one components

if (!vis[i]) { } i

dfs(node, parent) {

vis[node] = 1

for (auto it : adj[node]) {

if (!vis[it]) {

if (dfs(it, node)) return true; == true

}

else if (it != parent) return true;

}

}

return false;

}

}

SC → O(n)

Time → O(N + 2E) + O(n)

WORD LADDER-2

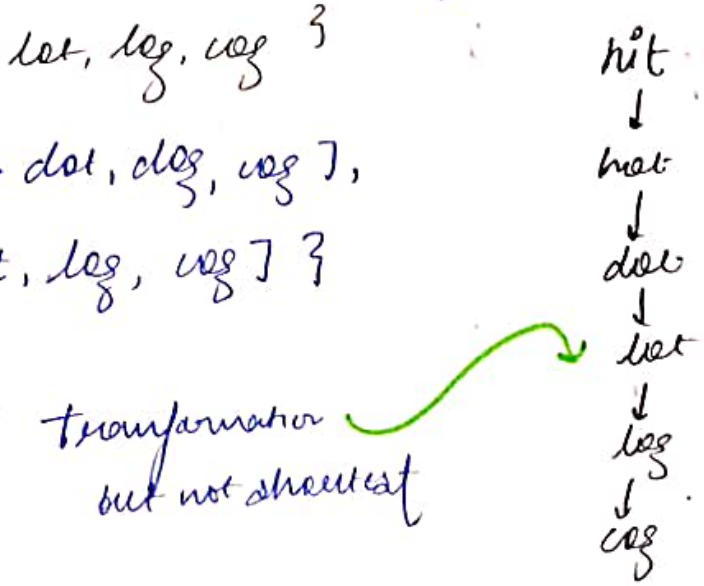
returning list of all shortest possible sequence

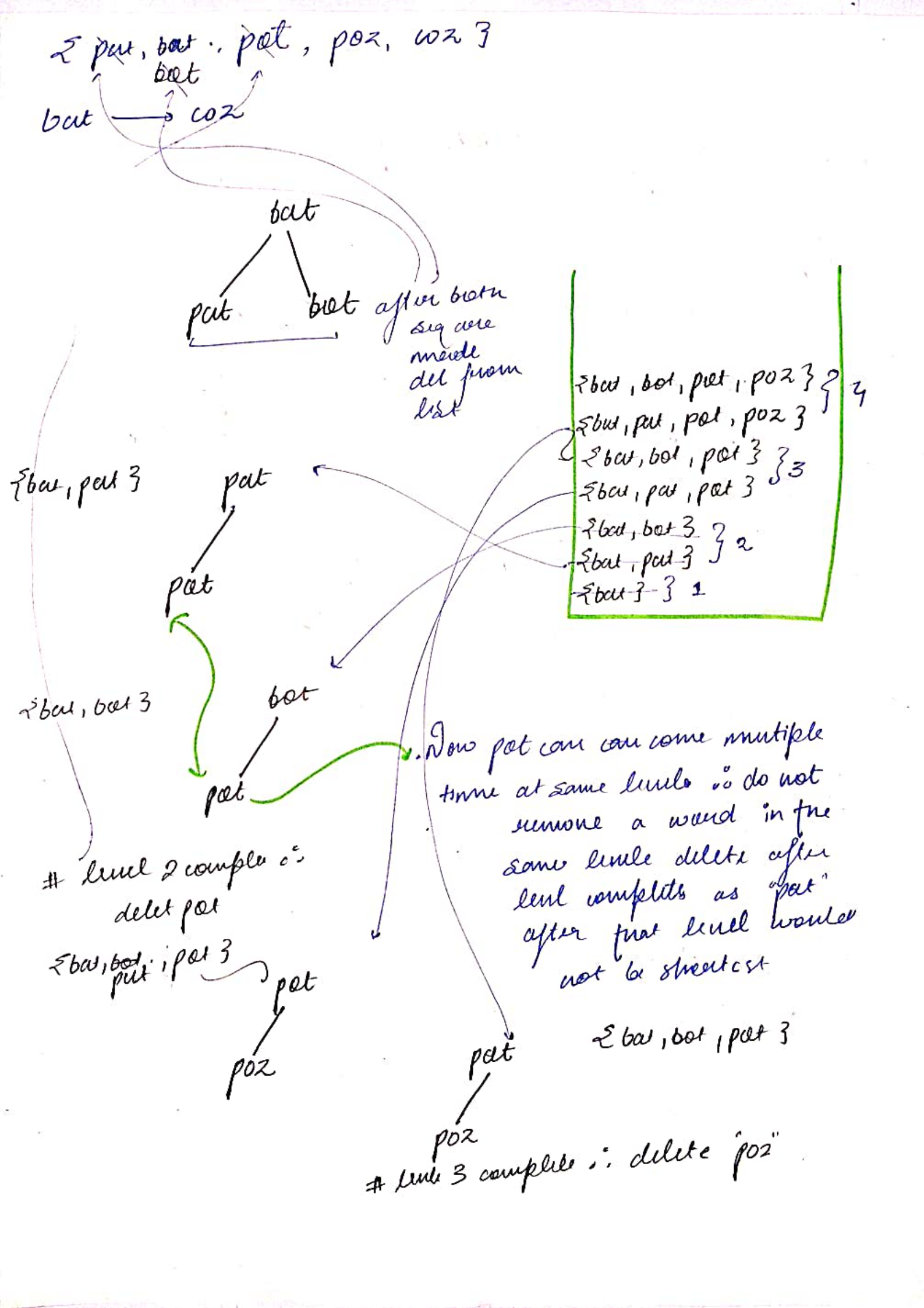
hit

cog

{ hot, dog, dot, lot, log, cog }

→ { [hit, hot, dot, dog, cog],
[hit, hot, lot, log, cog] }

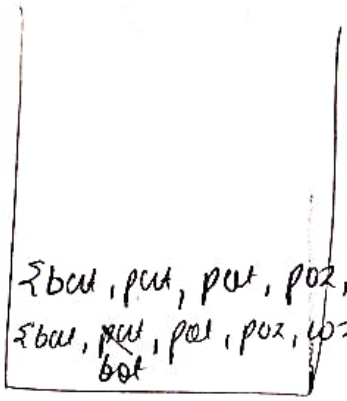



{ pat, bat, pat, poz, coz }
bat
bat → coz
bat
pat
bat
after both sig are made del from list
{bat, bot, pat, poz } 4
{bat, pat, pat, poz }
{bat, bot, pat } 3
{bat, pat, pat }
{bat, bot } 2
{bat, pat }
{bat } 1
{bat, pat }
pat
pat
{bat, bot }
bot
pat
Now pat can can come multiple time at same level ∴ do not remove a word in the same level delete after level completes as "pat" after that level words not to streetcat
level 2 complete ∴ delet pat
{bat, bot, pat }
pat
poz
{bat, bot, pat }
pat
poz
level 3 complete ∴ delete "poz"

{bat, bat, pat, pot, po2}

po2

co2



{bat, pat, pot, po2

po2

co2

limit reached ∴ delete "co2"

{bat, bat, pot, po2, (co2)} ↷ end word ∴ store in list

{bat, pot pot, po2, co2} ↷ end word ∴ store in list

>5

↺ when other items in a queue are taken and results in seq of >5 ∴ break as this will be next shortest