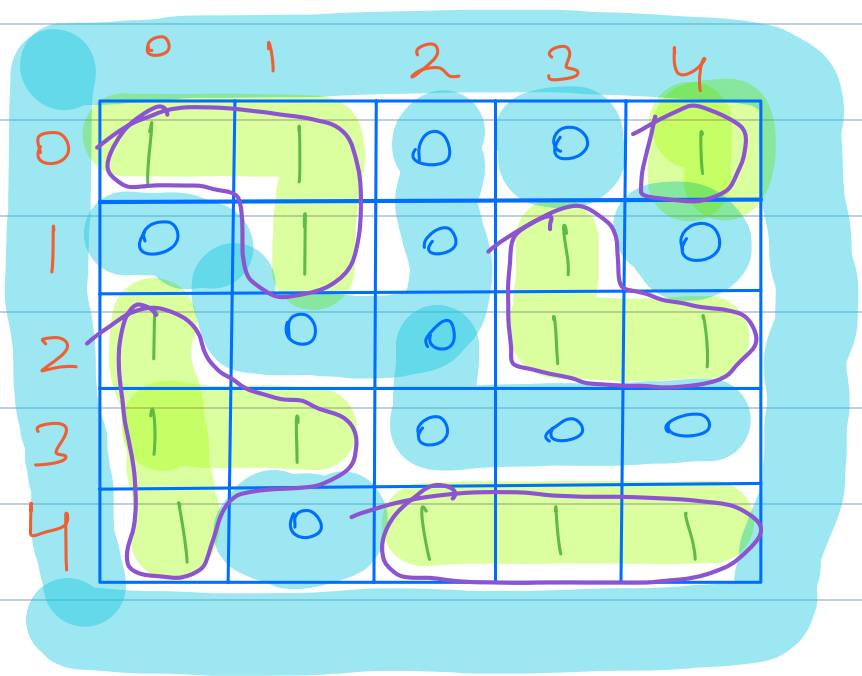
Today's Agenda:-

1. Number of Islands
2. Topological Ordering
3. Disjoint Set Union (D.S.U.)
4. Applications of DSU.

Starting 7:05

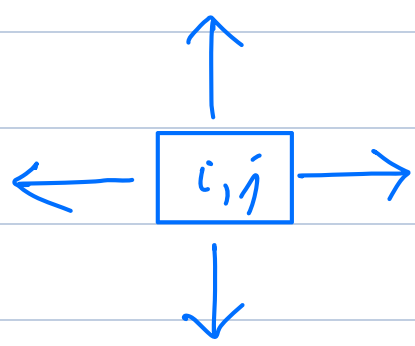
Number of Islands

N x M



1 - land

0 - water.



Every cell with '1' consider a node in your graph.

∴ No. of islands = Total No. of Connected Components.

Idea- From every unvisited cell which is land '1', call DFS / BFS.

Code:

```
boolean visited[N][M]; count = 0;

for (i=0; i<N; i++) {
    for (j=0; j<m; j++) {
        if (ar[i][j] == '1'
            && visited[i][j] == false)
        {
            dfs(ar, i, j, visited);
            count++;
        }
    }
}

return count;
```

x+1
y+0

x-1
y+0

x+0
y-1

x+0
y+1

3

3

```
void dfs(int[][] arr, int x, int y,
         bool[][] visited) {

    visited[x][y] = true;

    dx = {-1, 0, 1, 0}                    0,0.
    dy = {0, -1, 0, 1}

    for (k = 0; k < 4; k++) {

        ni = x + dx[k];
        nj = y + dy[k];

        if (ni >= 0 && ni < N &&
            nj >= 0 && nj < m &&
            arr[ni][nj] == 1 &&
            visited[ni][nj] == false) {

            dfs(arr, ni, nj, visited);

        }
    }
}
```

N*m.

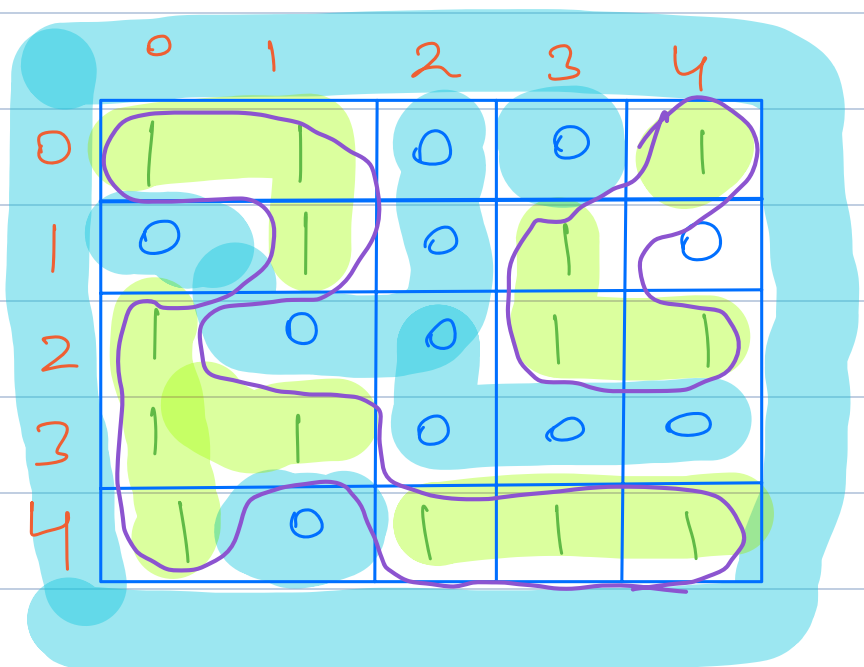
T.C → O(N+E).

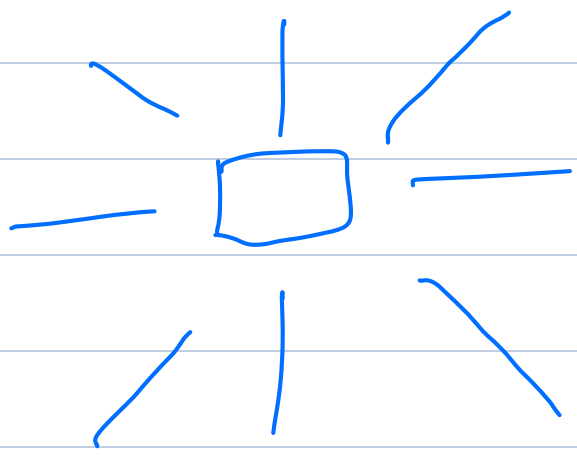
↓

N*m

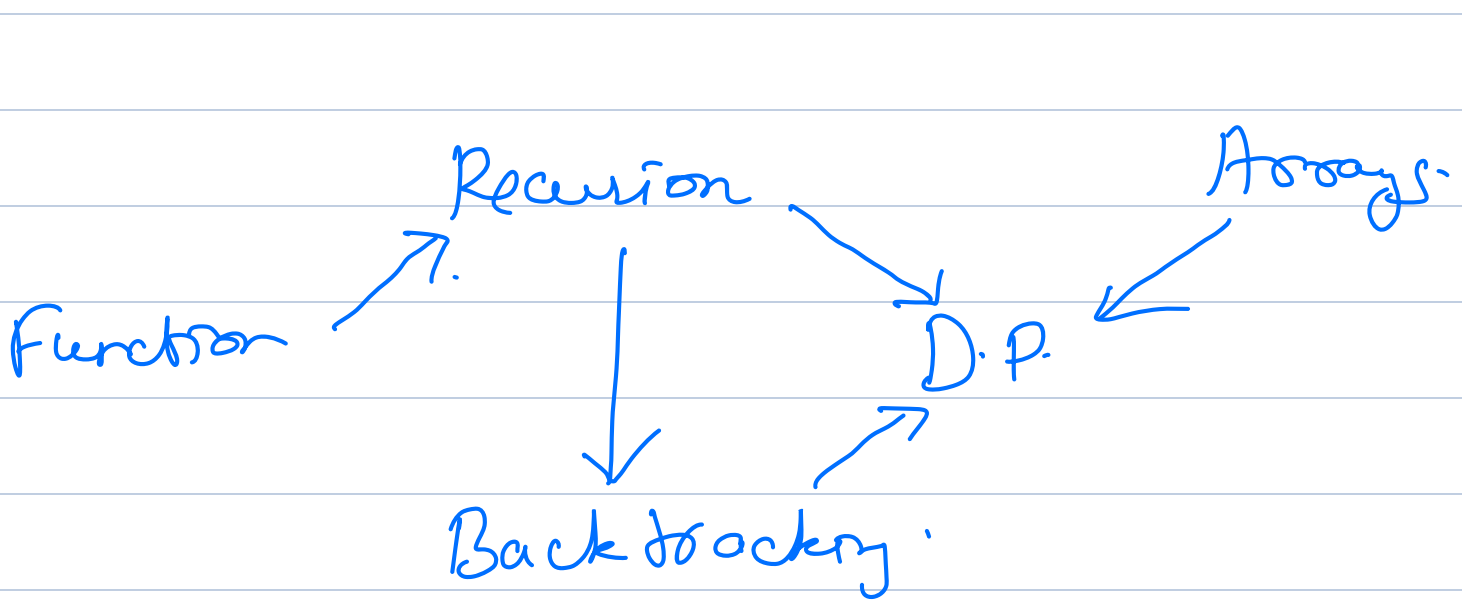
T.C → O(N*m).

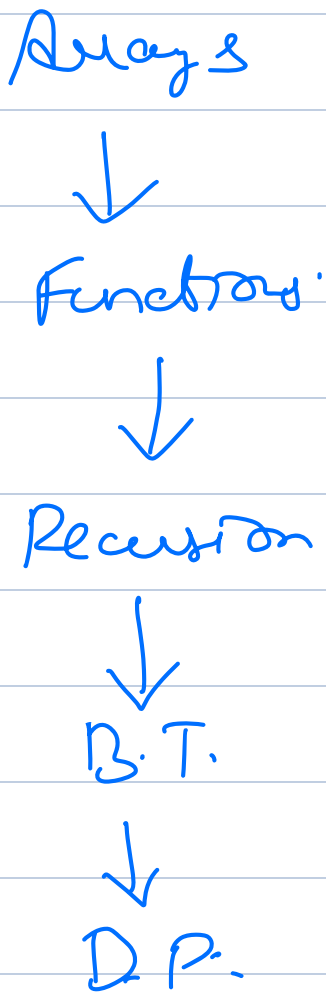
S.C → O(N*m)





Modify the dx & dy arrays to include 8 directions.





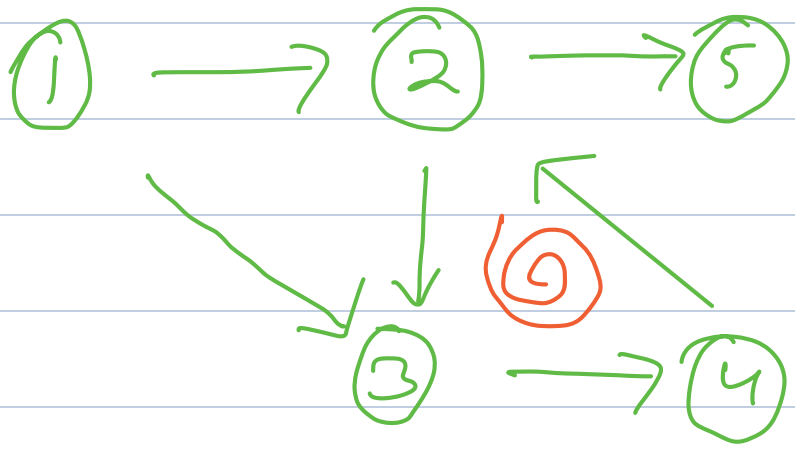
Q:- Given N courses with prerequisite of each course. Check if it is possible to finish all the courses.

N = 5

If x is a prerequisite of y.

$x \rightarrow y$

1 → 2, 3
2 → 3, 5
3 → 4
4 → 2



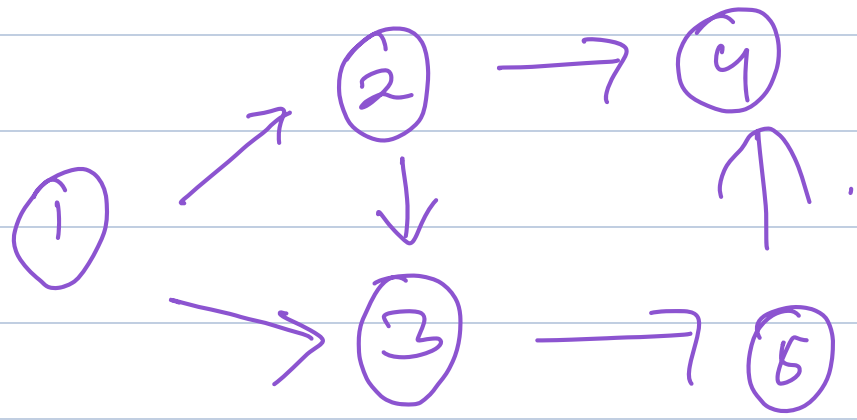
1, 3, 4, 2, 5. X.

1, 2, 3, 5, 4. X

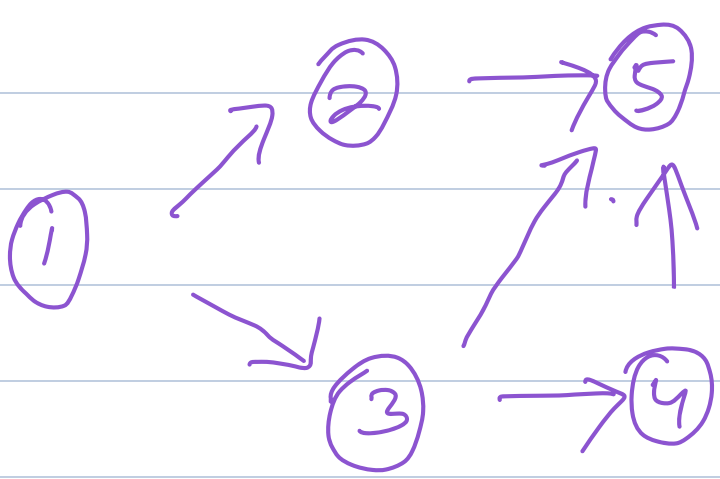
In any order, I'll not be able to complete all the cocurses.

Detect cycle in my graph.

Q:- Cycle doesn't exist. Identify the order..



1, 2, 3, 5, 4 .



1, 2, 3, 4, 5 ✓

1, 3, 4, 2, 5 ✓

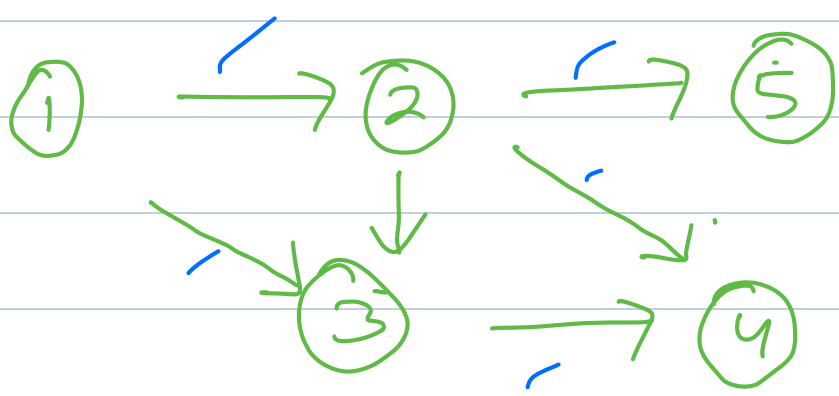
1, 3, 2, 4, 5 ✓

Multiple such ordering possible.

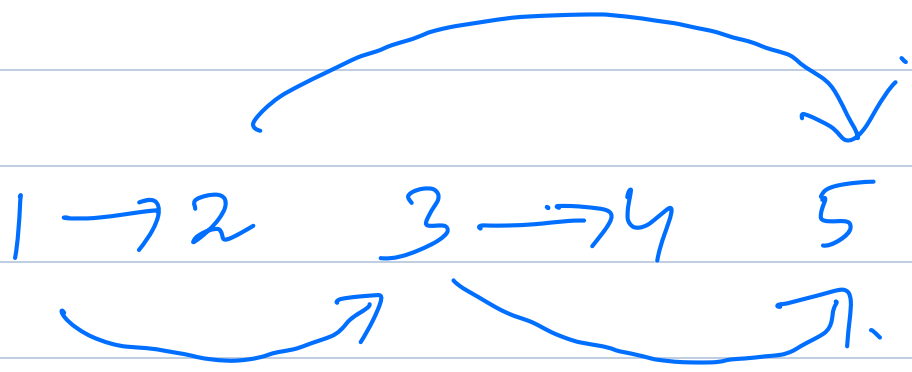
Topological Order / Sorting

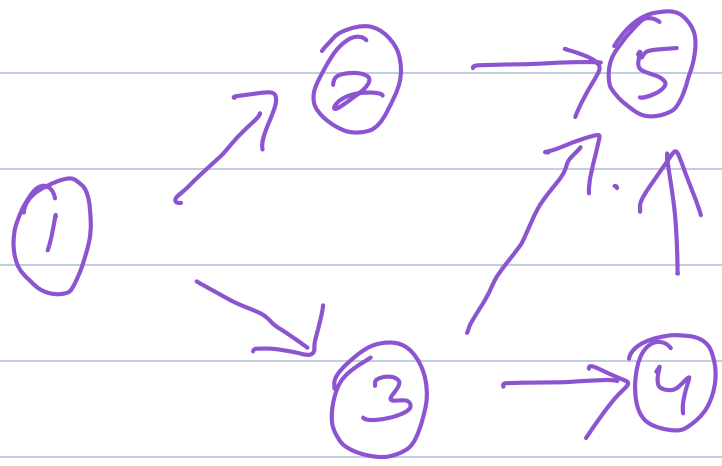
Linear order of nodes such that if there is an edge from i to j, then i should be present on l.h.s. of j.

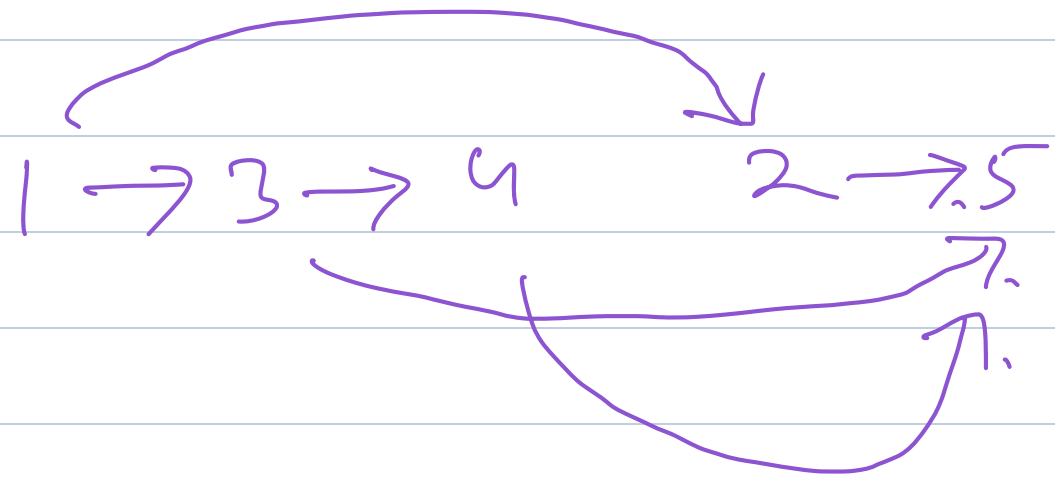
Only applicable for DAG.

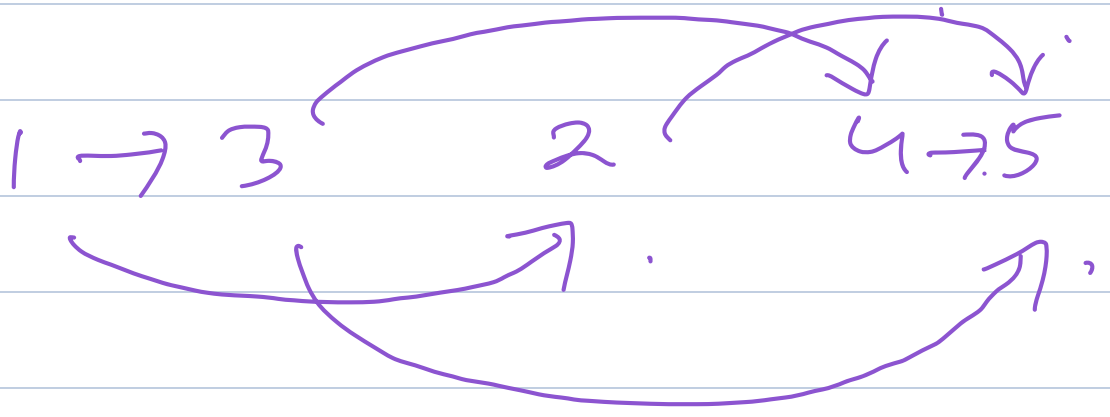


1 2 3 4. 5

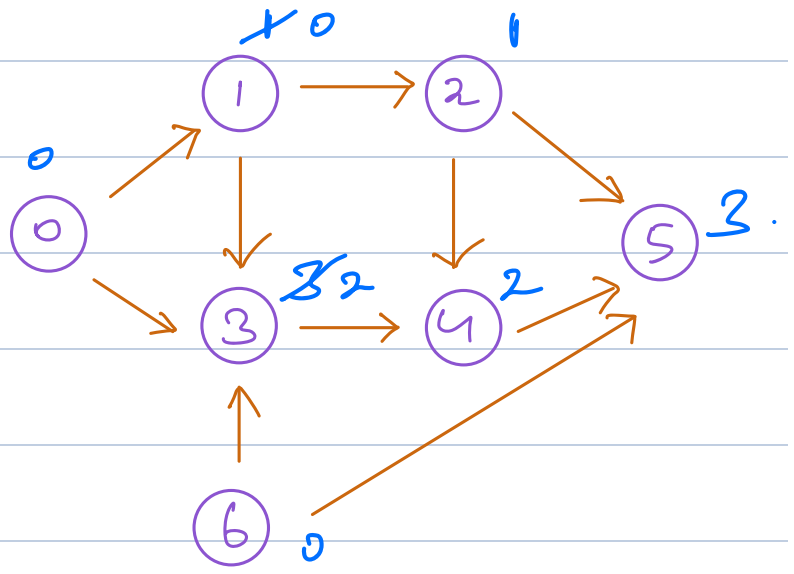








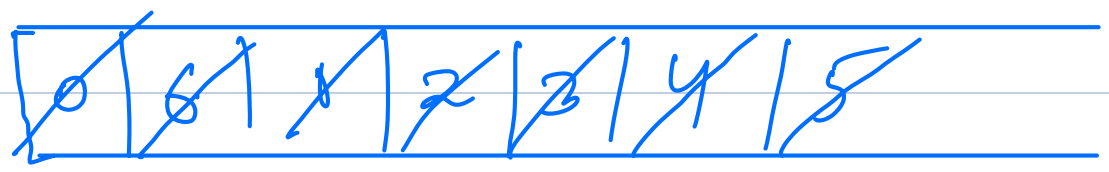
Find Topological Order



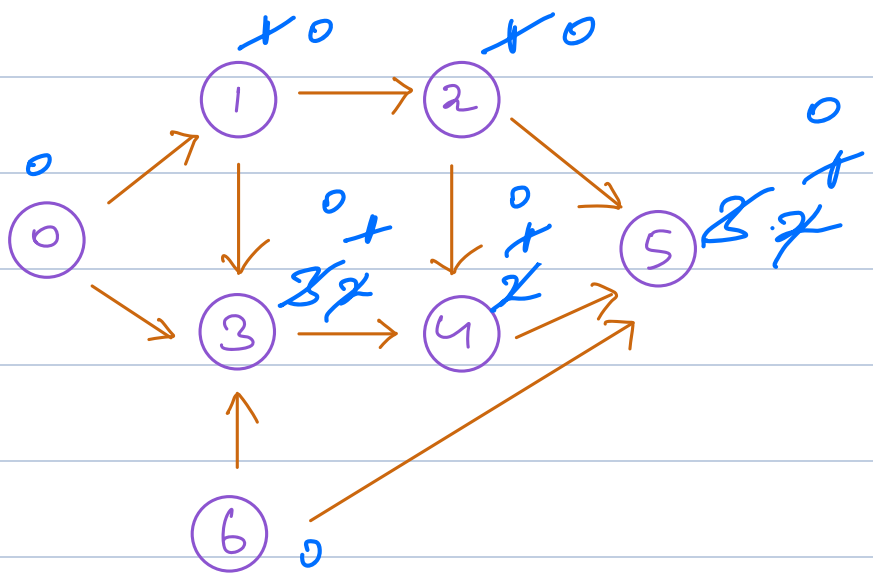
Step 1 :-
Find Indegree of each node.

```
n(N) ; ∀i in(i) = 0;

for(i=0; i<N; i++){
|   for(int nbr: graph(i)){
|     n[nbr]++;
|   }
}
```



0 6 1 2 3 4 5



Step 2:- Insert all the nodes with indegree 0 into a queue.

```
Queue <int> q;
for (int i=0; i<N; i++) {
    if (in[i] == 0) {
        q.insert(i);
    }
}
```

Step 3:- Start dequing and reduce the indegree of neighbouring nodes by 1. If at any point of time, indegree → 0, push it into the queue.

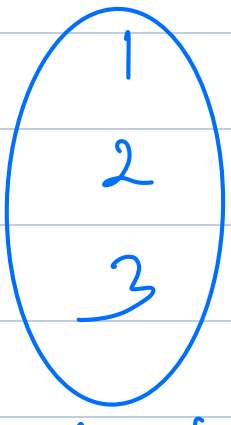
```
while (q.isEmpty() == false) {
    x = q.dequeue(); print(x);
    for (int nbr : graph[x]) {
        in[nbr] -= 1;
        if (in[nbr] == 0) {
            q.insert(nbr);
        }
    }
}
```

8:32.

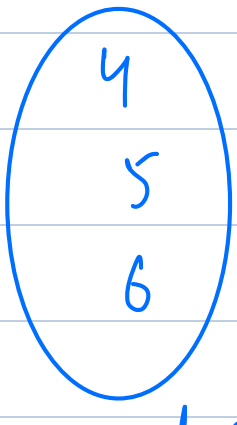
T.C → O(N+E)

S.C → O(N)

Disjoint Set Union



Set 1



set 2.

$S_1 \cap S_2 - \{\}$

$S_1 \cup S_2$

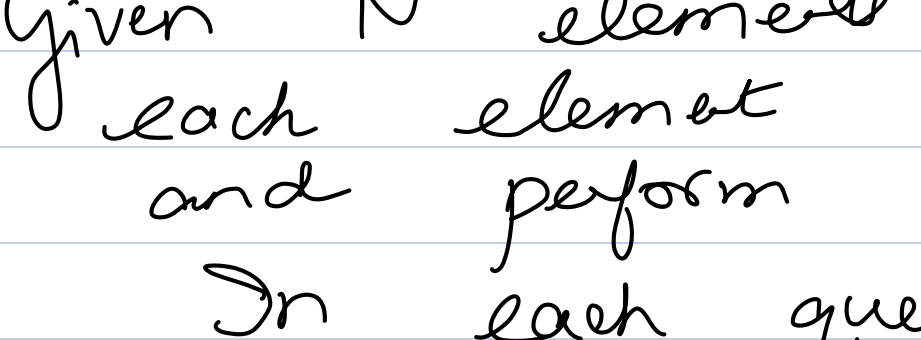
↳ {1, 2, 3, 4, 5, 6}.

Q:- Given N elements. Consider each element as a enique set and perform multiple queries. In each query, check if (u, v) belong to different sets.

If yes → merge the 2 sets & return true.

No → return false.

N = 4.



1 - {}
2 - {}
3 - {}
4 - {}

X

Queries

(1, 2) - true.
(3, 4) - true.
(1, 2) - false.
(1, 3) - true
(2, 3) - false.

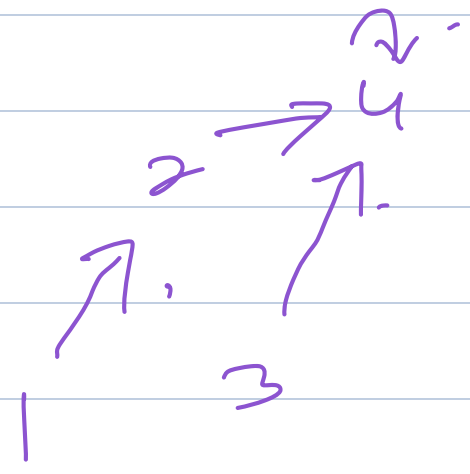
Idea:- Consider every element as a tree where where the node of a tree point to the parent node. & The root of that tree points to itself..

par[] - { 2 4 4 4 }.
1 2 3 4

Queries:

(1,2) - true.
(3,4) - true.
(1,2). - false.

(1,4) - true.



root(1) = 2
root(4) = 4.

root(1). x = 1.

x = 2.

x = parent(2); = 4.

(

```
boolean union(int x, int y) {
    rx = root(x);
    ry = root(y);
    if(rx == ry) {
        return false;
    }
    else {
        parent[rx] = ry
        //
        parent[ry] = rx;
        return true;
    }
}

int root(int x) {
    while(parent[x] != x) {
        x = parent[x];
    }
    return x;
}
```

O(Ht. of tree).

↓

O(N);

S.C → O(N);

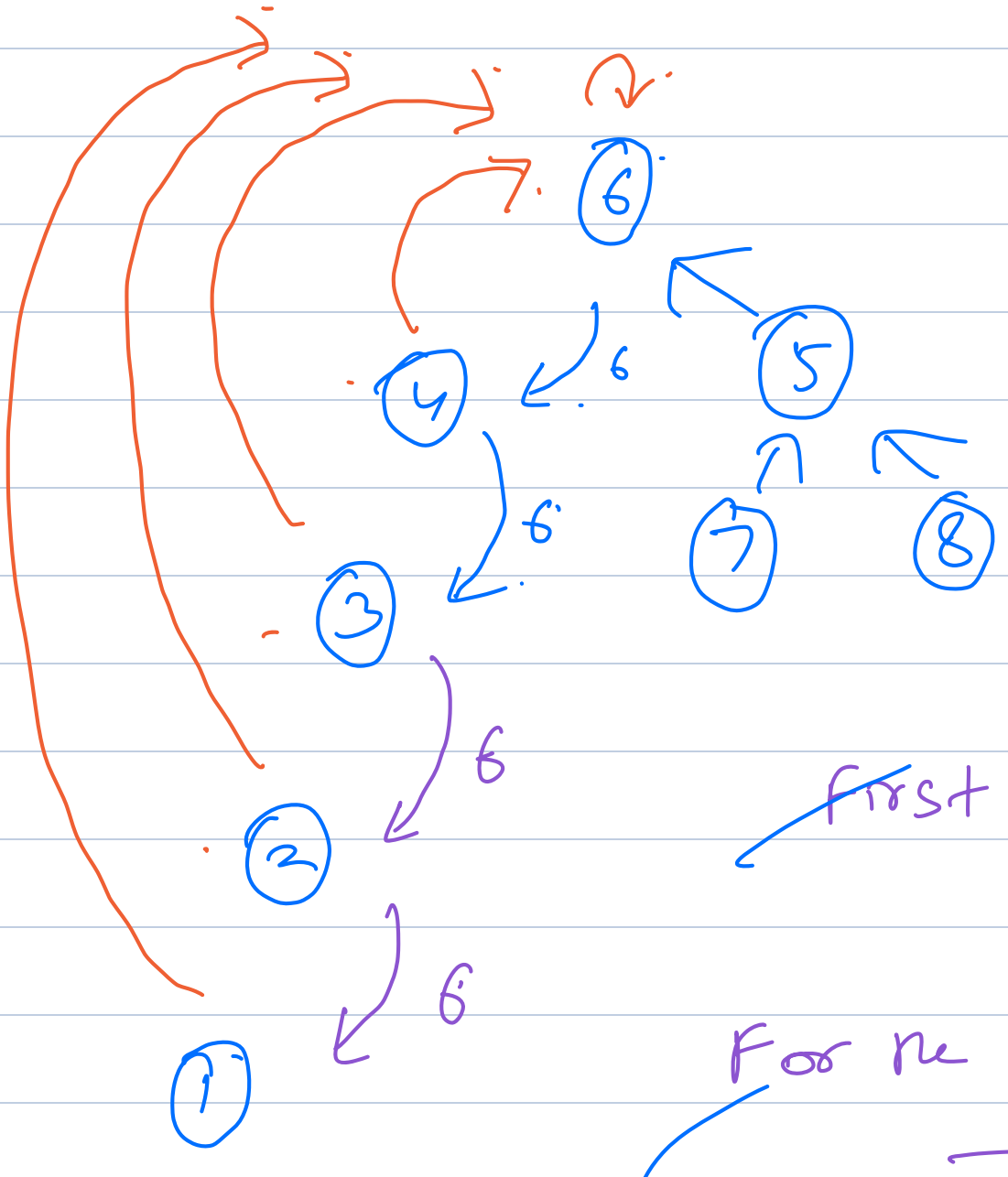
Optimise DSU.

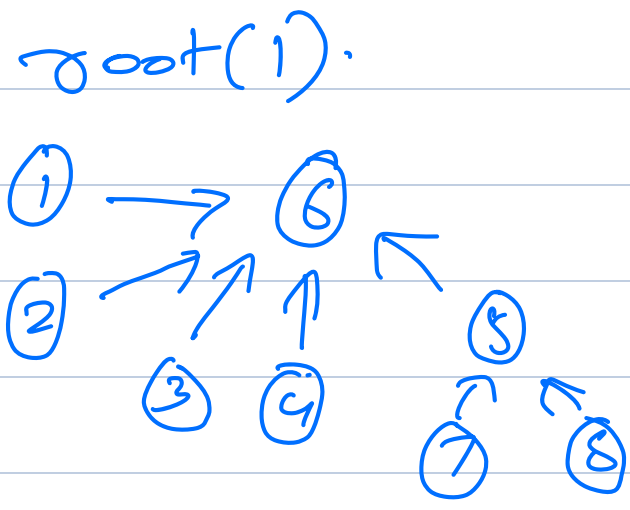
→ Union By Rank. → H.W.
($log_2 N$)

→ Path Compression

O(1) Amortized.

Path Compression.





~~first~~ time

→ K iterations.

For the next K calls,

→ 1 iteration.

For K+1 calls → K + K.
iterations

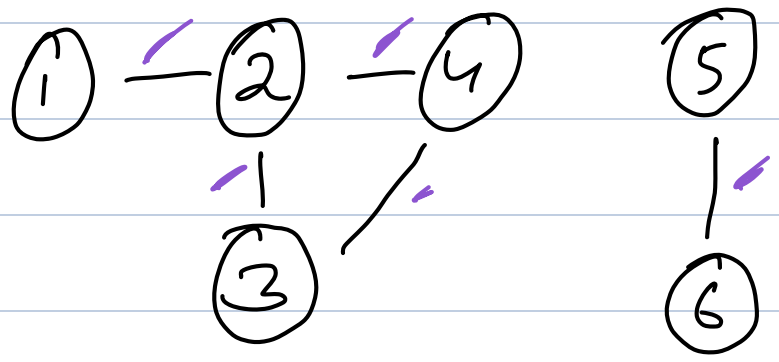
1 call. → $\frac{2k}{(k+1)}$

```
int root(N x){
    if(x == par[x]) {return x;}
    o = root(par[x]);
    par[x] = o;
    return o;
}
```

T.C. → O(1) Amortised :-

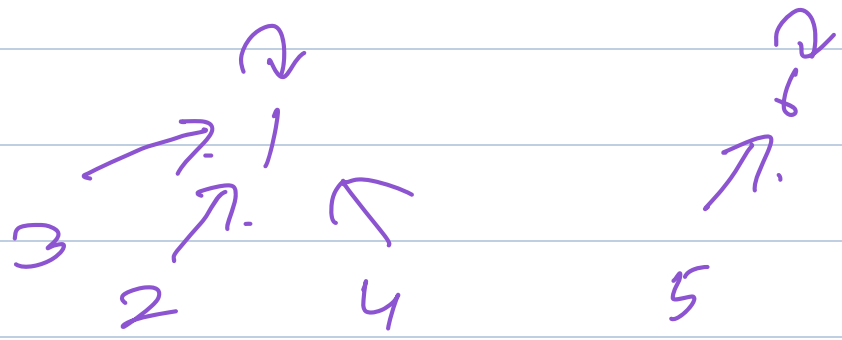
Applications of DSU

(1) Check if an undirected graph is cyclic or not.

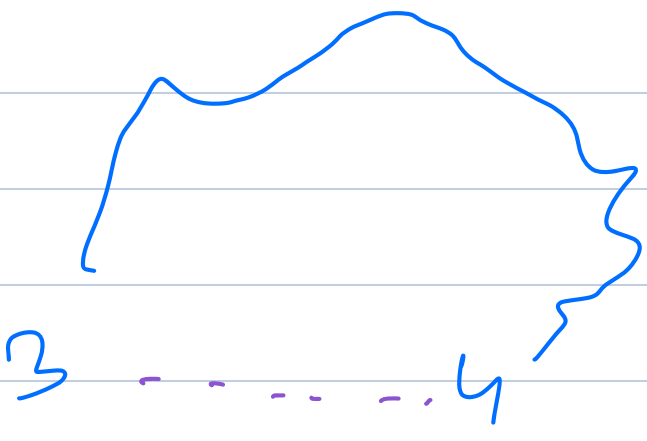


→ For all nodes, consider them as independent sets.

→ ∀ edges, take a union (u, v).

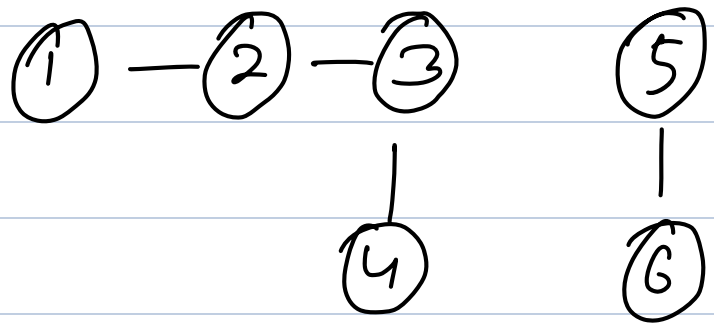


1 2 → true.
2 4 → true.
1 3 → true
2 3 → true
3 4 → false. → A cycle exists.
5 6 → true.



Iterate on all the edges of my graph. If union (u, v) == false, that means cycle is detected..

2. Check if a graph is connected or not.



→ For all nodes, consider them as independent sets.

→ ∀ edges, take a vertex (u, v).

→

If root nodes, 2 than I can say graph is disconnected..

(3) Minimum Spanning Tree.

↳ Prim's

↳ Kruskal.

→ Dijkstra.