Graphs - 2

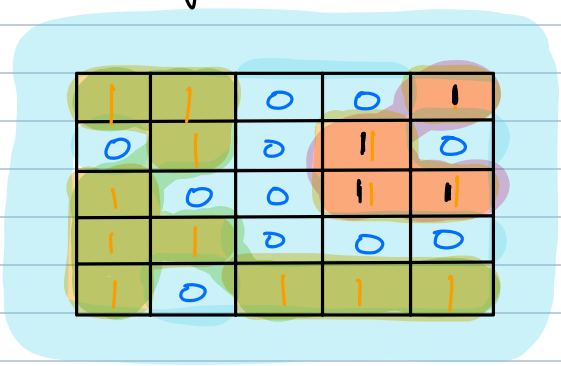
Topics To Cover:

1. No. of Islands II
2. Topological Order → 2 methods
3. DSU [Disjoint Set Union]
4. Application for DSU.

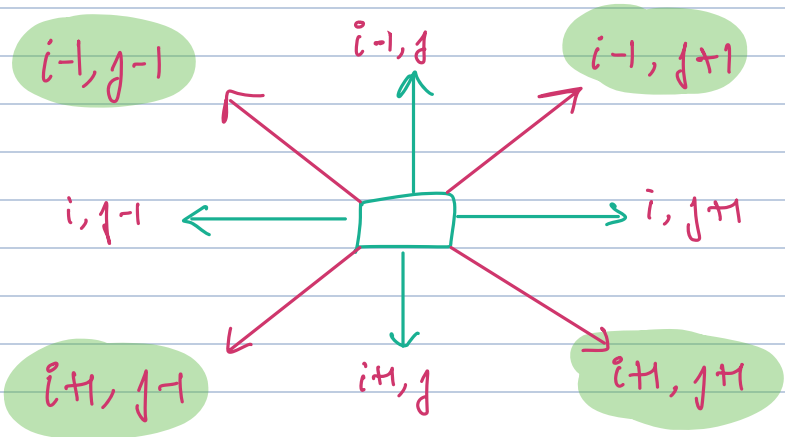
Song: Radio Gaga - Queen

Hi Everyone!!!

Number of Islands with Diagnol movement allowed

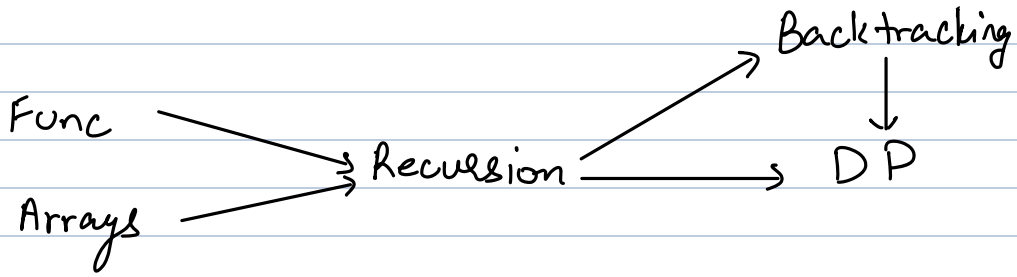


Ans: 2




```
islands = 0

for (i=0 ; i<N ; i++)
    for (j=0 ; j<M ; j++)
        if (Map[i][j] == 1)
            islands++;
            DFS (Map, i, j)
```

```
fn { DFS ( Map , i , j )
    Map [i] [j] = -1 ;

    dx = [ 0   1   0   -1   +1   -1   -1   +1 ]
    dy = [ 1   0   -1   0   +1   -1   +1   -1 ]
                                 diagnols

    for { ( k = 0 ; k < 8 ; k++ )
        Ni = i + dx [k] ;
        Nj = j + dy [k] ;
        if ( Ni ≥ 0 && Nj ≥ 0 && Ni < N && Nj < M
             && Map [Ni] [Nj] == 1 )
        {
            DFS ( Map , Ni , Nj ) ;
        }
    }
}
```

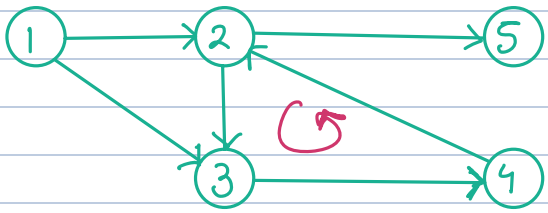


Q1. Given N courses with pre-requiste of each course. Check if it is possible to finish all the course.

N = 5

$x \rightarrow y$ (x is a pre-requiste of y)

1 → {2, 3}
2 → {3, 5}
3 → {4}
4 → {2}



Ans : False

∴ If there exists a cycle then we can surely say we cannot learn all the subjects.

If cyclic ⟶ False

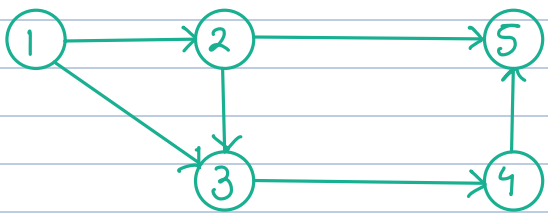
else ⟶ True

idea 1 : follow path and see if cycle exists (Graph-1)

TC : $\Theta(V+E)$

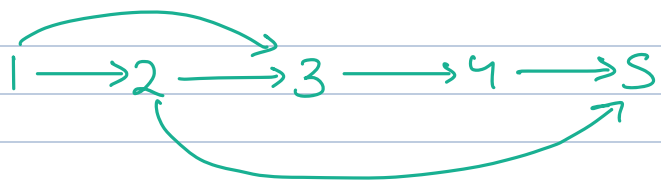
Topological order / sort

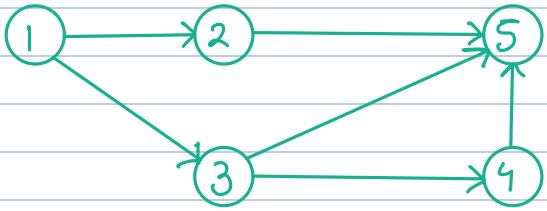
Linear ordering of nodes such that if there is an edge from $i \rightarrow j$ then i should be present on LHS of j



∴ If we are able to put it in a topological order ↩ Acyclic

else its cyclic



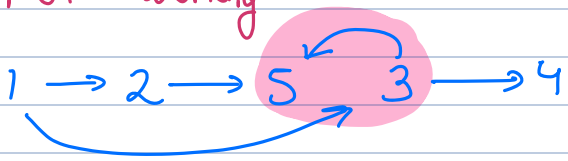


① 1 → 2 → 3 → 4 → 5

② 1 → 3 → 2 → 4 → 5

③ 1 → 3 → 4 → 2 → 5

Not working

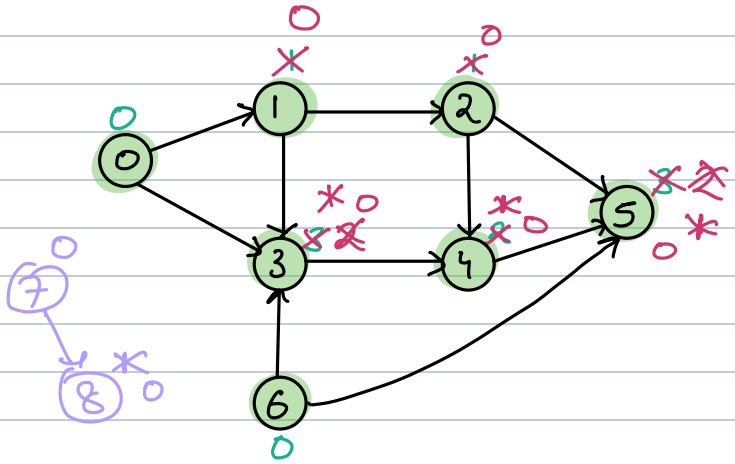


* Topological order can have multiple outcomes.

① In / Out Degree

graph[]
0 → {1, 3}
1 → {3, 2}
3 → {4}
6 → {3, 5}
4 → {5}
2 → {4, 5}

index

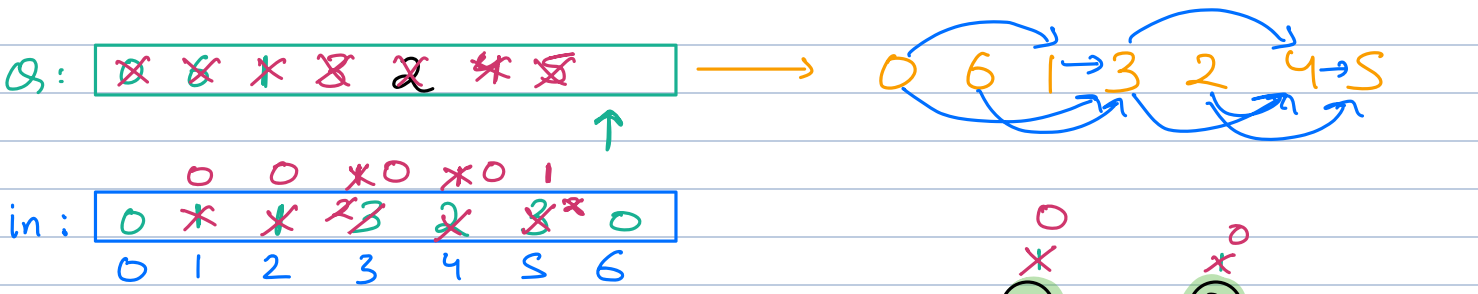


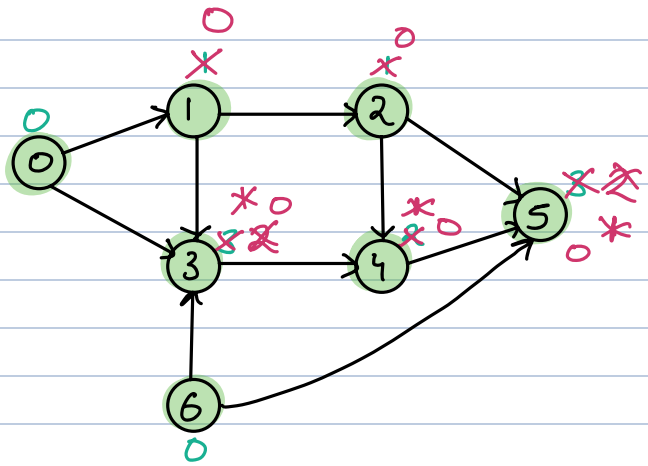
Step 1: Calc indegree.

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

```
int [N+1] in;

for (i = 0; i <= N; i++)
    for (int nbr : graph[i])
        in[nbr]++;
```

Step 2: Insert all Nodes with in order = 0, and then continue Calc.



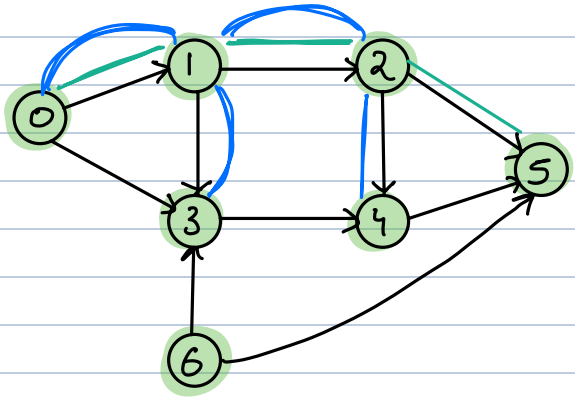


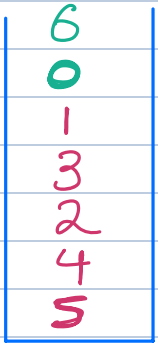

```
Queue <int> Q;

for(i=0; i ≤ N; i++)
{
    if(in[i] == 0)
    {
        print(i);
        Q.insert(i);
    }
}

while(Q.size > 0)
{
    temp = Q.poll; ⟶ Q.pop();

    for(int nbr : graph[temp])
    {
        in[nbr]--;

        if(in[nbr] == 0) ⟶ Node has no dependency left.
        {
            Q.insert(nbr);
            print(nbr);
        }
    }
}
```

TC: $O(V+E)$
SC: $O(V)$

Break: 10:37 – 10:45

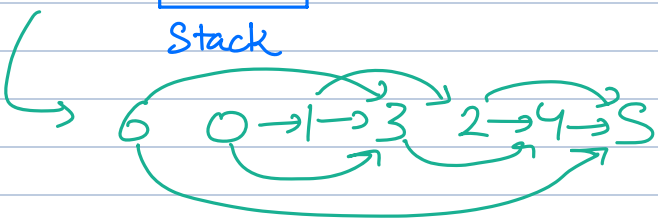
② DFS + Stack





Stack

★ DFS ② if going back to prev call add node to stack.




```
boolean visited [N+1] ;
Stack <int> st;

for ( i=0 ; i ≤ N ; i++)
{
    if ( visited [i] == False)
        DFS ( graph, visited, i, st )
}

while ( st.isEmpty () == False)
{
    print ( st.poll / st.pop) ;
}

fn DFS ( graph, visited, i, st)
{
    visited [i] = True ;

    for ( int nbr : graph [i])
    {
        if ( visited [nbr] == False)
        {
            DFS ( graph, visited, nbr, st)
        }
    }

    st.push (i)
}
```

TC: $O(V+E)$
SC: $O(V)$

Disjoint Set Union:

S1 ∩ S2 → { } (Intersection)

S1 ∪ S2 → {1, 2, 3, 4, 5, 6} (Union)

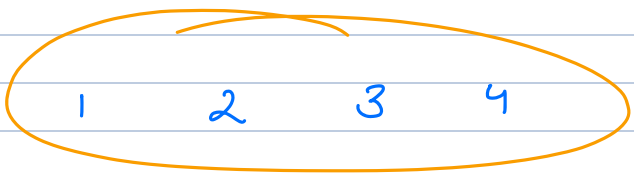
Q. Given N ele, consider each ele as a unique set and perform multiple queries.

In each query check if (u, v) belong to different sets

If Yes → Merge and return True

If No → return False.

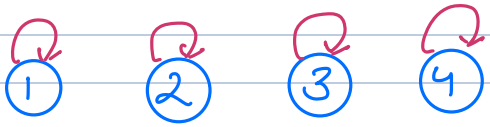
N = 4



Q:
(1, 2) → True
(3, 4) → True
(1, 2) → False
(1, 3) → True
(2, 3) → False

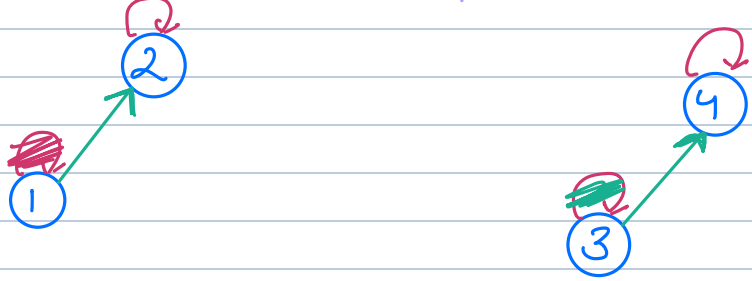
* Every Node is a self appointed parent of itself.

N = 4



par[1] = 2 or par[2] = 1

Q:
(1, 2) ✓
(3, 4) ✓
(1, 2) → False
(1, 3)
(2, 3)

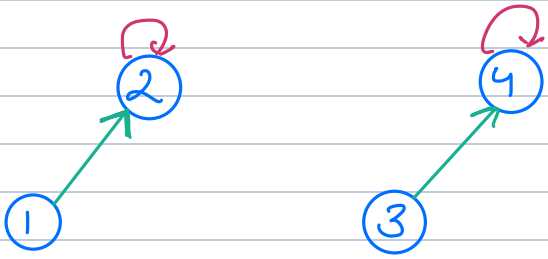


par[3] = 4

(1, 3) → root(1) → 2

root(3) → 4

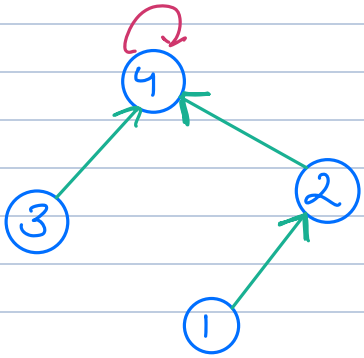
∴ Do not belong to the same set.



par[3] = 2

* We only change root nodes and not child nodes

✓ par[2] = 4 or par[4] = 2



par:

0	2	4	4	4
0	1	2	3	4

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

TC: Θ(ht. of the tree)

$\Theta(V)$

TC with queries:

$\Theta(Q * V)$

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