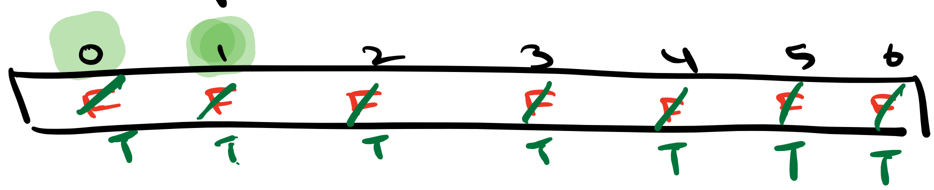
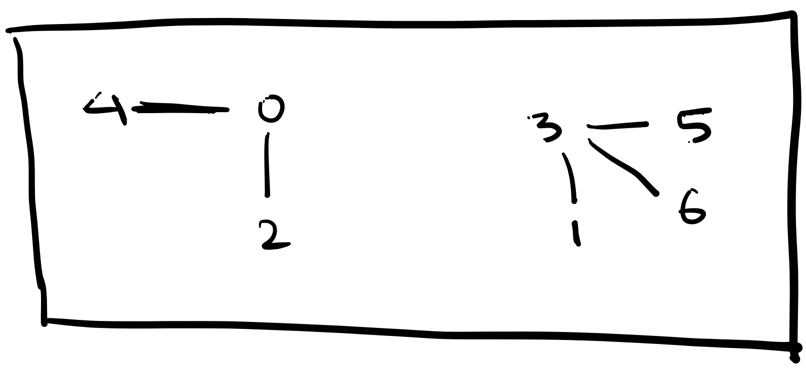
Agenda

1. No. of Islands
2. Topological Sort
3. Union - Find

Traverse from every node



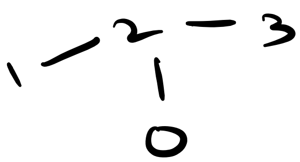
Undirected graphs



N = 7

No. of connected components = 2

N = 4



Connected graph

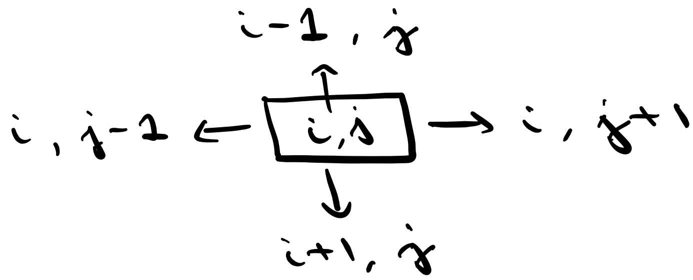
↓

Every node can be reached from every other node.

No. of connected components = 1

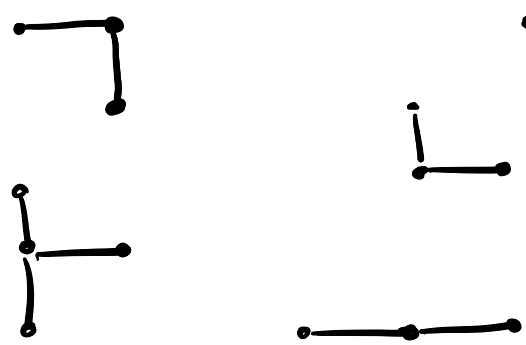
1. Given a mat[ ][ ] of 1 (representing land) and 0 (representing water), find no. of islands.

Island → chunk of land with water in all directions



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

ans = 5



How a matrix represents a graph?

① Every cell is a node

② ← • → At max 4 cells are neighbours

mat[5][5] → vis[5][5] = <F>

	0	1	2	3	4
0	~~1~~ 2	~~1~~ 2	0	0	~~1~~ 2
1	0	~~1~~ 2	0	~~1~~ 2	0
2	2 ~~1~~	0	0	2 ~~1~~	2 ~~1~~
3	~~1~~ 2	~~1~~ 2	0	0	0
4	~~1~~ 2	0	~~1~~ 2	~~1~~ 2	~~1~~ 2

cnt = $\emptyset$ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

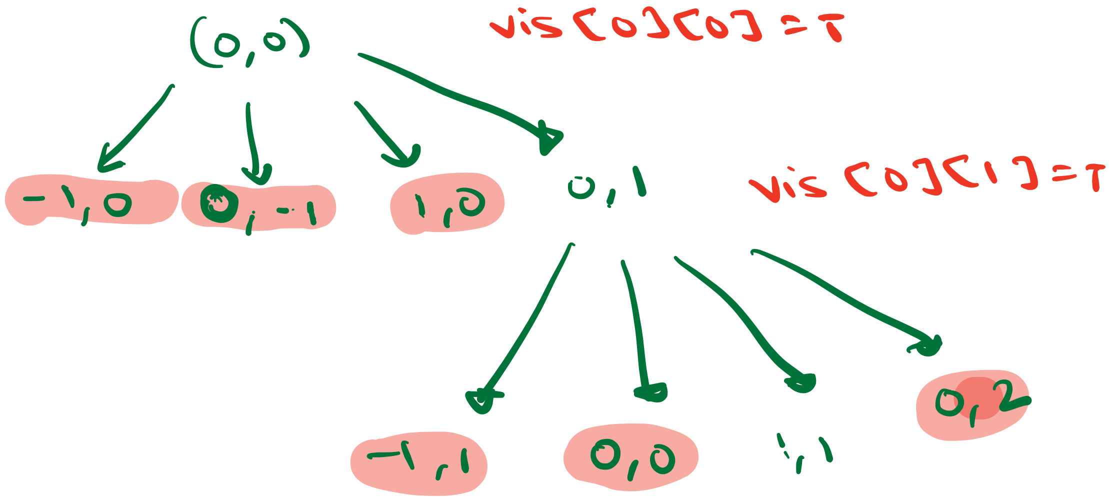
No. of islands

↓

piece of land

↓

dfs from every land cell



```
int island (int mat [N][M]) {

    int cnt = 0
    for (i=0 ; i< N ; i++)
     for (j=0 ; j<m ; j++) {
         if (mat [i][j] == 1) {
             cnt++
             dfs (i, j, mat)
         }
     }
}
```


```
void dfs (int i, int j, int mat[][]) {
  if( i<0 || i≥N || j<0 || j≥m ||
      mat[i][j] == 0 || mat[i][j]==2)
            return;

      mat[i][j] = 2

      dfs (i-1, j, mat)
      dfs (i, j-1, mat)
      dfs (i+1, j, mat)
      dfs (i, j+1, mat)
}
```

No. of nodes = N*M

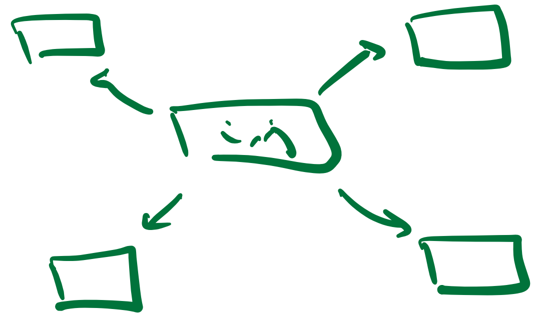
No. of edges = 4*N*M

TC: O(N+E) = O(N*M + 4*N*M)

= O(N*M)

SC: O(N*M)

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

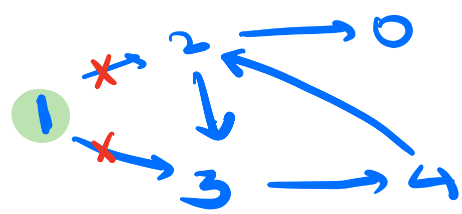


2. Given N courses with prerequisites, check if it is possible to finish all courses.

Input N = 5 0, 1, 2, 3, 4

x is a prerequisite of y

x	y
1	2
1	3
2	3
2	0
3	4
4	2

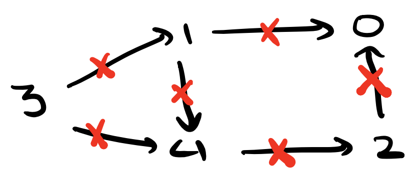


3 → 4 → 2

If cycle
return false
else
return true

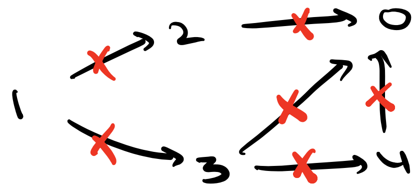
Topological sort

3 1 4 2 0

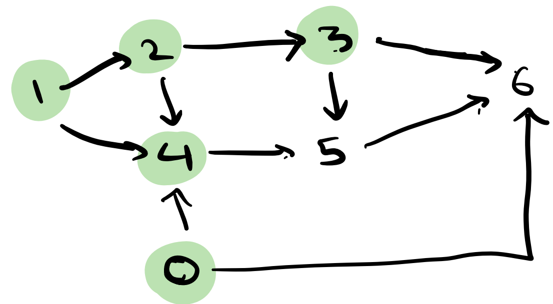


Linear ordering of nodes s.t. if there is an edge from i to j; i will be before/on left of j

→ 1 2 3 4 0
→ 1 3 2 4 0
→ 1 3 4 2 0



incoming edges
↓
indegree



~~0~~	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~

N = 7
indeg [7]

0	1	2	3	4	5	6
0	0	~~1~~	~~1~~	~~3~~	~~2~~	~~3~~
		0	0	~~2~~ ~~1~~ 0	~~1~~ 0	~~2~~ ~~1~~ 0

0 1 2 3 4 5 6

```
// N, M
   u  v                          u ⟶ v

  int  indeg [N]
list<int>  adj [N]

  for (i=0 ; i< M; i++) {
        // u  v
        adj[u]. insert (v)
        indeg [v] ++
  }

 queue <int>  q

 for (i=0 ; i<N ; i++) {
      if (indegree [i] ==0)
             q.enqueue (i)
 }
```

```
int cnt = 0
while (q.size() > 0) {

    int c = q.front()
    q.dequeue()
    print (c)                cnt++
    for (int i=0; i<adj[c].size() ; i++){
        int nbr = adj[c][i]
        indegree[nbr] --

        if (indegree[nbr] == 0) {
            q.enqueue (nbr)
        }
    }
}
```

O(N+E)

TC : O(V+E)

SC : O(N/V)

// check indegree of every node

if indeg[any node] > 0

$\Rightarrow$ cycle

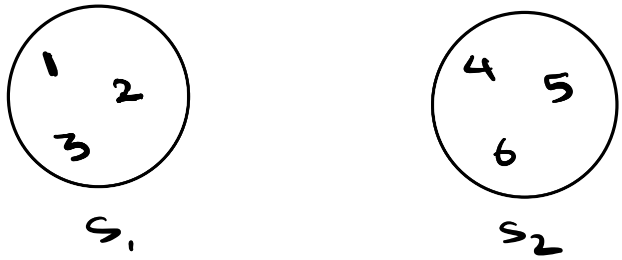
if (cnt == N)

$\rightarrow$ no cycle

else

$\rightarrow$ cycle

Disjoint Set Union (DSU)

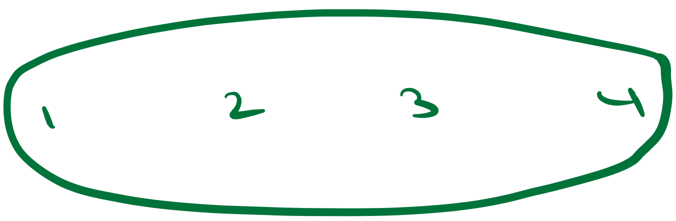


$S_1 \cap S_2 = \phi$ (Nothing)

$S_1 \cup S_2 = \{1,2,3,4,5,6\}$

1. Given N elements, consider each element as a unique set & perform multiple queries. In each query if (u,v) belong to different sets, we do their union & return true, otherwise false.

N=4



Queries

(1,2) True

(3,4) True

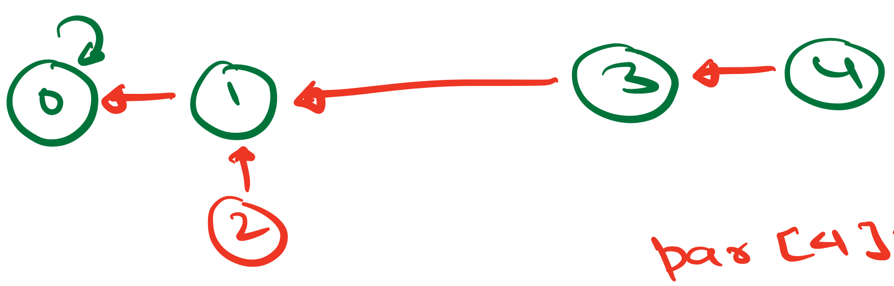
(1,2) False

(1,4) True

(2,3) False

	0	1	2	3	4
par	0	0	1	1	3

N=5



par[4] = 3

root(4)

Queries

(1,2) T

(3,4) T

(1,4) T

(2,4) F

r(2)=1 r(4)=1

(1,3) F

1 1

(4,0) T

1 0

TC: O(N)

```
bool union (x, y) {

    rootx = find (x)
    rooty = find (y)
    if (rootx == rooty) {
        return false
    }

    else {
        if (rooty > rootx)
            par [rooty] = rootx
        else
            par [rootx] = rooty
    }

    return true
}
```

union (3, 10)
1 8

```
int find(x, par[]){
    if(x != par[x]){
        return find(par[x], par)
    }
    return x
}
```

x

4

$\boxed{3}$

TC: O(N)

↑0

find(4)

↓↑0

find(3)

↓↑0

find(1)

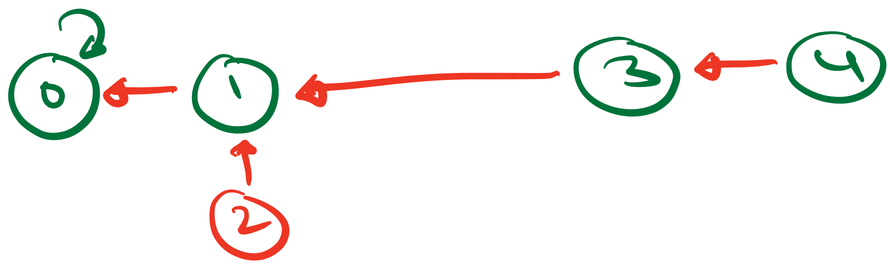
↓↑0

find(0)

par

0	1	2	3	4
0	0	1	1	3

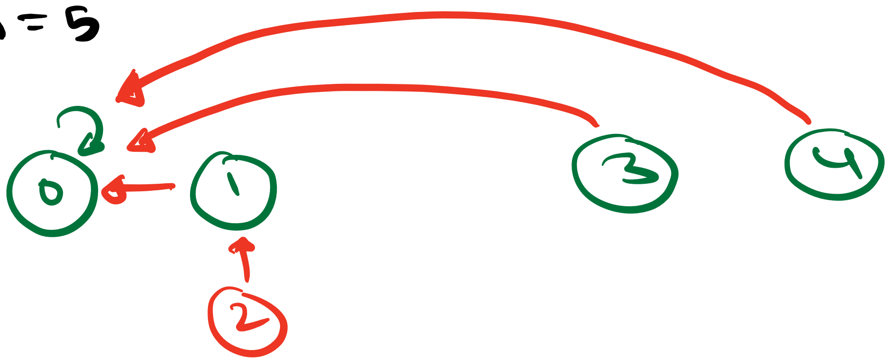
N=5



par

0	1	2	3	4
0	0	1	1	3

N=5



Path Compression

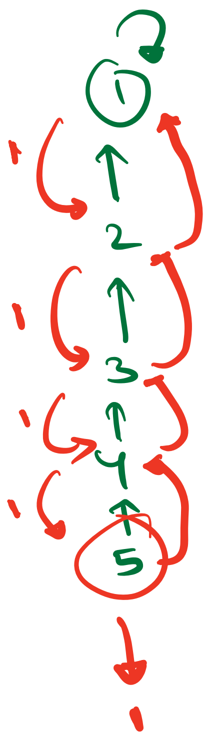
→ TC: O(1) avg

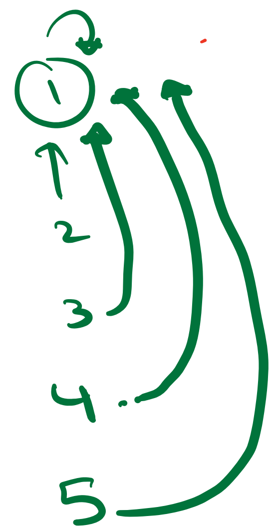
```
int find (x, par[]) {
    if (x != par[x]) {
        par[x] = find(par[x], par)
        return par[x]
    }
    return x
}
```

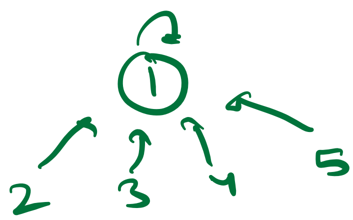
x
4
$\boxed{3}$

find(4)
1
3

union() → TC: O(1)







find(5) → 5 iter
next (5) → 1 iter

Amortized TC: O(1)