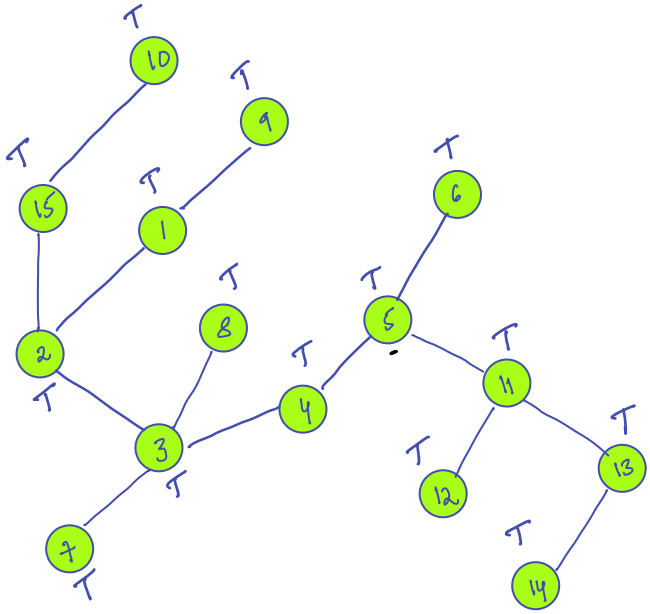
Todays Content:

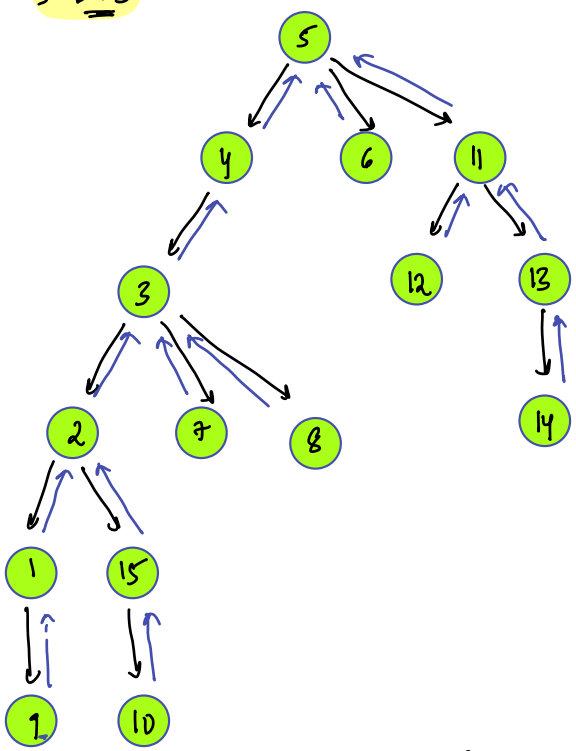
- → DFS
- → No: of Connected Components
- → Cycle detection in Undirected graph
- → No: of Islands
- → Multi Source BFS

DFS: S - 5, D - 14

↳ Depth First Search:



→ DFS .



Pseudocode:

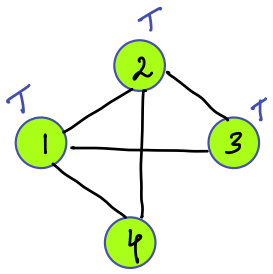
↳ given graph Input → Adj List - List<int> g[N+1] ← no. of nodes

```
void dfs( List<int> g[], bool vis[], int s) {
    if (vis[s] == True) { return }
    vis[s] = True // s is getting visited 1st time
    // We apply dfs on dfs n its connected nodes
    for(i=0; i< g[s].size(); i++) { // g[s] is list, ith index
        v = g[s][i]
        dfs(g, vis, v)   // Even if vis[v] = true by calling it
    }                    // won't effect us, at start we anyy
}                        // check if it's already visited

main() {
    return vis[dest] } TC: O(N+E)   ← Edges
}
```

// Diff:

Eg1:



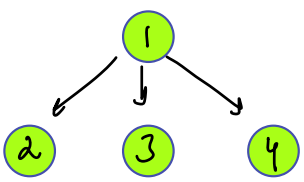
Q): un/directed graph shorter path from S → D : {BFS}

Q): un/directed graph S → D, if we can visit?

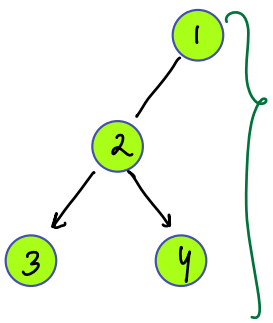
- {BFS}
- {DFS}

BFS : S : 1

0	1
x	2 3 4



DFS : S : 1



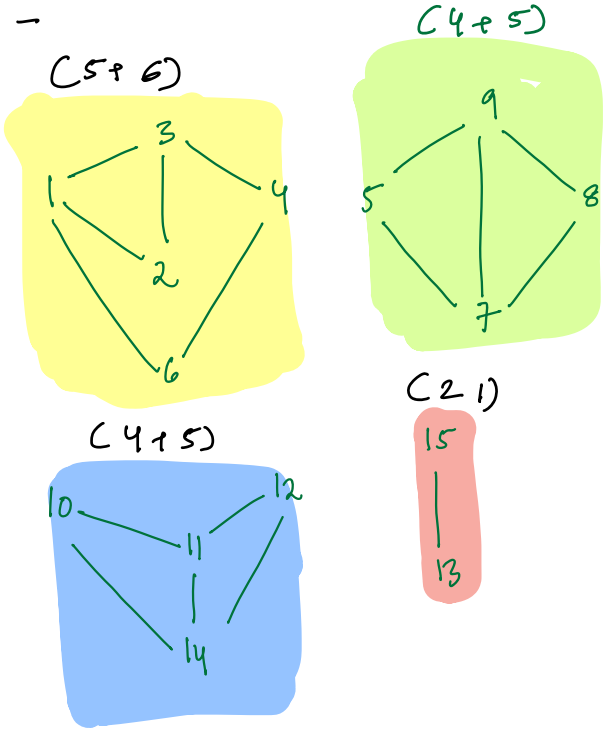
dist from 1 → 3 : 2

not shortest distance

20) Given {undirected graph} find no: of Connected Components?

A component is said to be connected, if from every node, we can visit all nodes, in component vis[16]

Eg: N=15 → ans=4



0	F		c = 0
1	F → T	DFS(1) / BFS(1)	c = c+1
2	F → T	×	
3	F → T	×	
4	F → T	×	
5	F → T	DFS(5) / BFS(5)	c = c+1
6	F → T	×	
7	F → T	×	
8	F → T	×	
9	F → T	×	
10	F → T	DFS(10) / BFS(10)	c = c+1
11	F → T	×	
12	F → T	×	
13	F → T	DFS(13) / BFS(13)	c = c+1
14	F → T	×	
15	F → T	×	c = 4

TC: O(N+E)

SC: O(N+E)

Pseudocode:

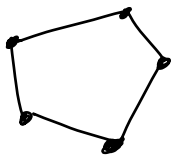
list<int> g[N+1] (Adj list)

bool vis[N+1]; (visited array)

```
c = 0
for(i = 1; i <= N; i = i+1){
    if(vis[i] == false){
        DFS(g, vis, i)
        c++
    }
}
return c;
```

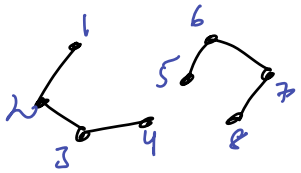
3(a) Cycle detection in Undirected graphs using Connected Comp

Eg1:



→ Cycle present

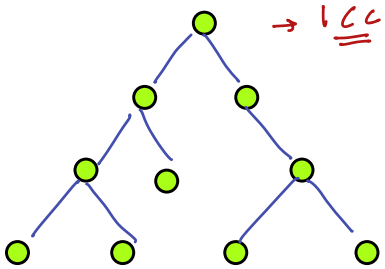
Eg2:



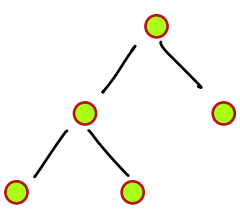
→ Cycles not there

obs1:

(Tree N Nodes, Edges N-1)



→ 1 cc



obs2:

// given undirected graph N Nodes

5

#Components

1

#Edges

== N-1 : {No Cycle}

> N-1 : {Cycle}

2

C_1 C_2

→ N_1 Nodes N_2 Nodes

→ N_1-1 N_2-1

→ Total Edges

$N_1 + N_2 - 2$ = $N-2$

= E = (N-2) : {No Cycle}

= E > (N-2) : {Cycle}

3

C_1 C_2 C_3

N_1 N_2 N_3

N_1-1 N_2-1 N_3-1

Total Edges = N-3

E = N-3 : {No Cycle}

E > N-3 : {Cycle}

Final Obs: Given undirected graph N Nodes & E Edges

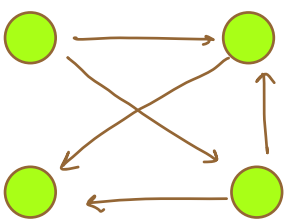
Nodes Edges → } Both In Inputs

C = No: of Components → TC: O(N+E)

```
If( E == N-C ){
|   // no cycle
}
else{
|   // There is cycle
}
```

} SC: O(1)

// Above Approach Directed graph as well?



→ N: 4

→ E: 5

→ {#Still No Cycle}

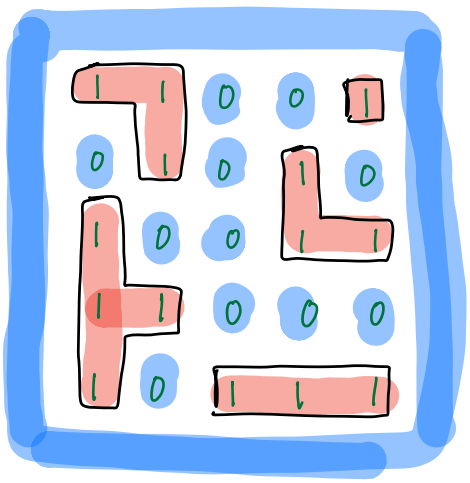
→ 10:40 PM

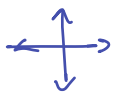
408) No of Islands

Given a matrix of 1's & 0's find no: of Islands are present?

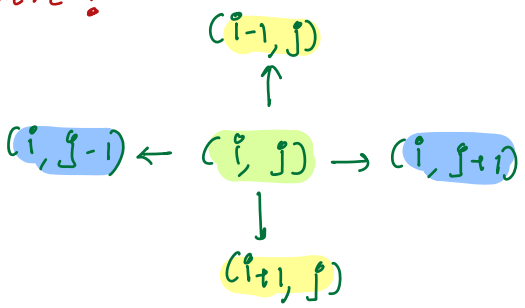
↳ Water surrounded in 4 directions

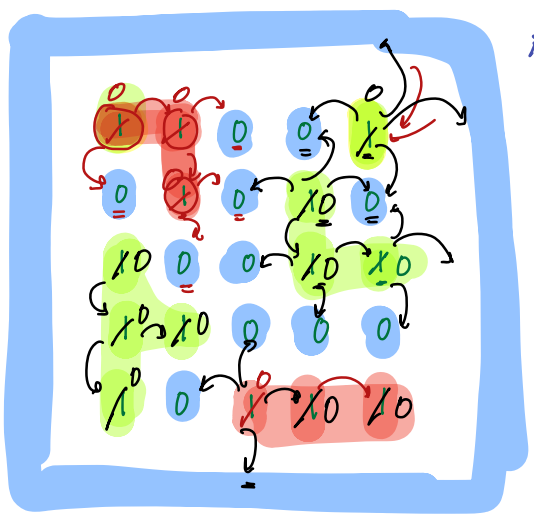
mat[N][M] → given matrix





→ 108) How many connected components are there?





How many times we need to apply DFS, to make all node → 0 : 5 times

// Pseudocode:

source cell

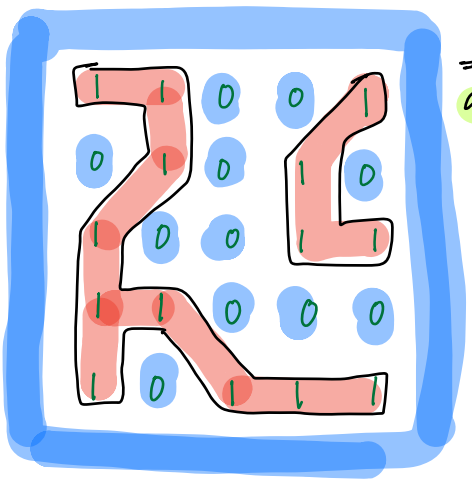
```
void dfs(int mat[][], int r, int c) {
    if (r<0 || c<0 || r>=N || c>=M || mat[r][c]==0) {
        return;
    }
    mat[r][c]=0 //visited  // Apply DFS to it's adj nodes
    dfs(mat, r-1, c)
    dfs(mat, r, c-1)
    dfs(mat, r+1, c)
    dfs(mat, r, c+1)
}
main(){
    // given mat[N][M]
    c=0;
    for(i=0; i<N; i++){
        for(j=0; j<M; j++){
            if(mat[i][j]==1){
                dfs(mat, i, j), c=c+1
            }
        }
    }
    return c;
}
```

408) **No of Islands**

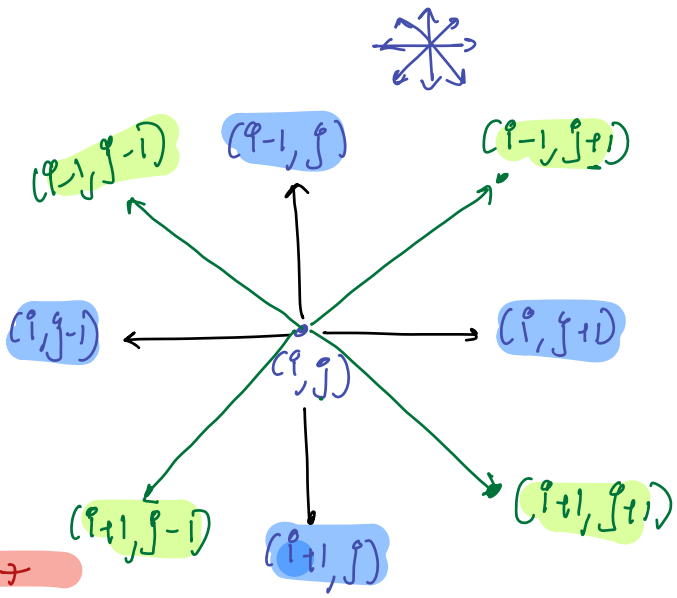
Given a matrix of 1's & 0's find no: of Islands are present?

↳ Water surrounded in 4 directions

mat[N][M] → given matrix



⇒ ans ⇒ 2



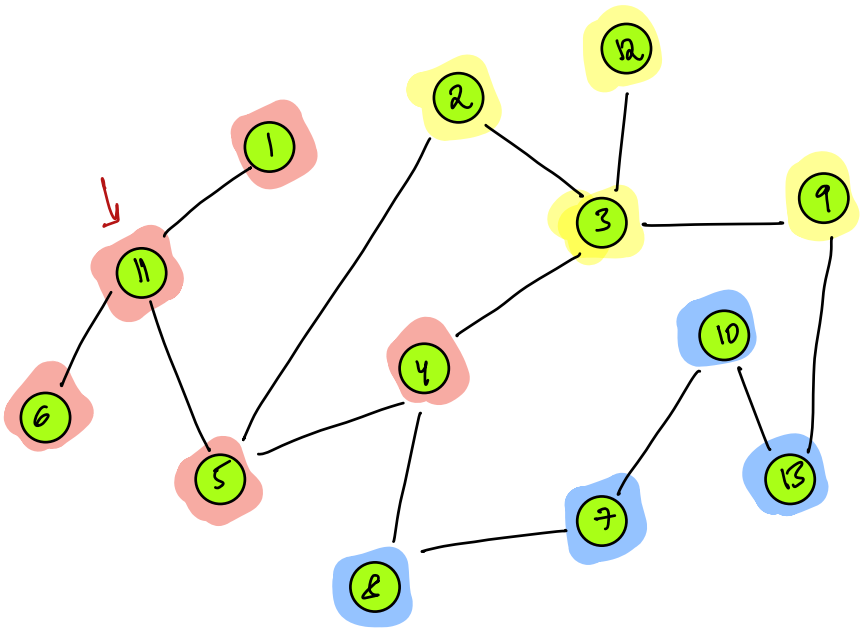
0 1 2 3 4 5 6 7

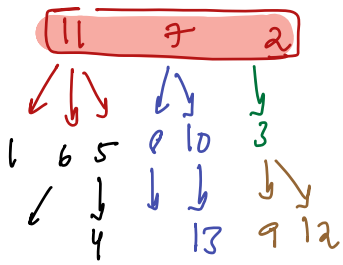
int x[] ⇒ {-1, -1, 0, 1, 1, 1, 0, -1}

int y[] ⇒ {0, 1, 1, 1, 0, -1, -1, -1}

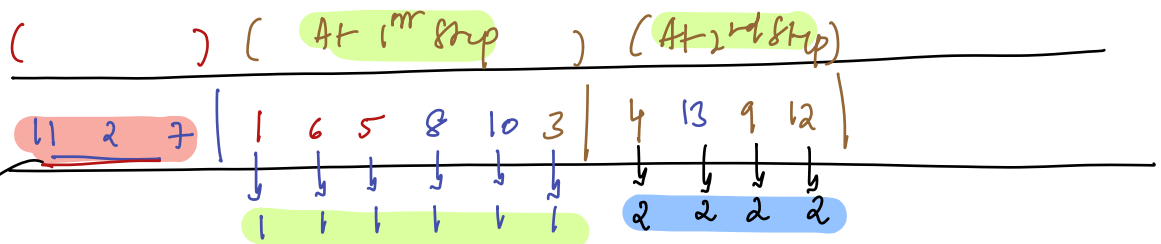
```
void dfs(int mat[][], int r, int c) {
    if (r<0 || c<0 || r>=N || c>=M || mat[r][c]==0) {
        return;
    }
    mat[r][c]=0 //visited // Apply DFS to it's adj nodes
    for(i=0; i<8; i++) {
        dfs(mat, r+x[i], c+y[i])
    }
}
```

58)





// Super Heros:



Multi Source BFS

1) Queue<int> q, dist[]

push all Source Nodes in q's } q { mark them visited }

Idea:

while (q.size()>0) {

int u = q.front(); q.remove();

{ Iterate & push all the non-visited connected nodes of u inside que }

}

Pseudocode:

```
bool BFS( list<int> g[N+1], list<int> s , int d ){
    bool v[N+1] = {F}
    int dis[N+1] = {N+1}
    Queue<int> q;
    for(i=0; i< s.size(); i++){
        q.insert(s[i]), vis[s[i]] = 0, dis[s[i]] = 0
    }
    while( q.size() > 0){
        int u = q.front()
        q.delete();
        // for node u we need to node connected to u?
        for( i=0; i< g[u].size(); i++){
            int v = g[u][i], dis[v] = dis[u]+1
            if( vis[v] == False){
                q.insert(v)
                vis[v] = True
            }
        }
    }
}
```

// About Pseudocode: