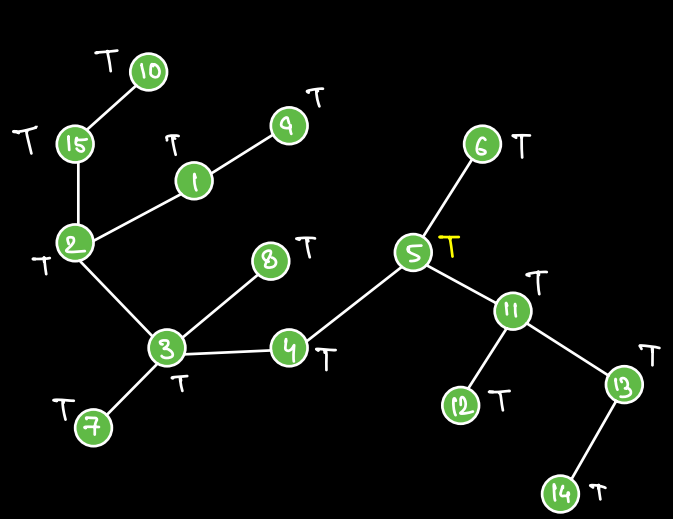
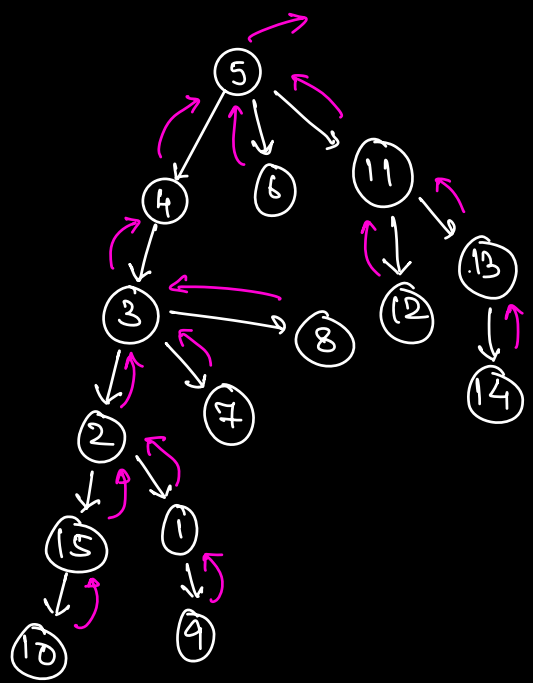
Dfs (Depth First Search) S=5, D=14






```
bool path(N, E, u[], v[], src, dest){
    list<int> g[N+1]; //Graphs-1 class.
    bool vis[N+1] = {false};
    dfs(g, N, E, src);
    return vis[dest];
}

void dfs(list<int> g, N, E, src){
    if(vis[src] == true) return;
    vis[src] = true;
    for(i=0; i< g[src].size(); i++){
        v = g[src][i];
        dfs(g, N, E, v);
    }
}
```

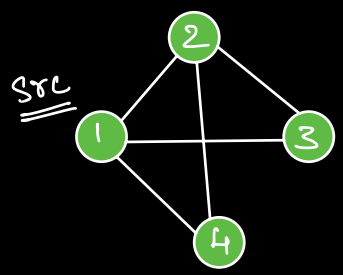
TC : O(E) {Same as BFS}

SC : O(E + N + N) ≈ O(E) {E >> N}

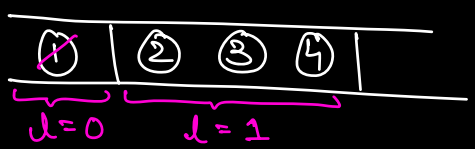
Stack, Visited[]

Adj List

Ex

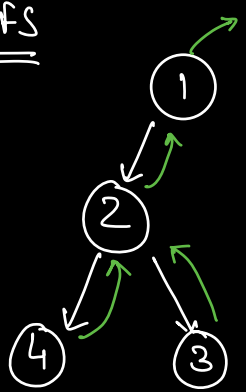


BFS



⇒ Shortest path of node 2, 3 & 4 from node ① is 1 only.

DFS



⇒ Length of path from ① to ④ = 2

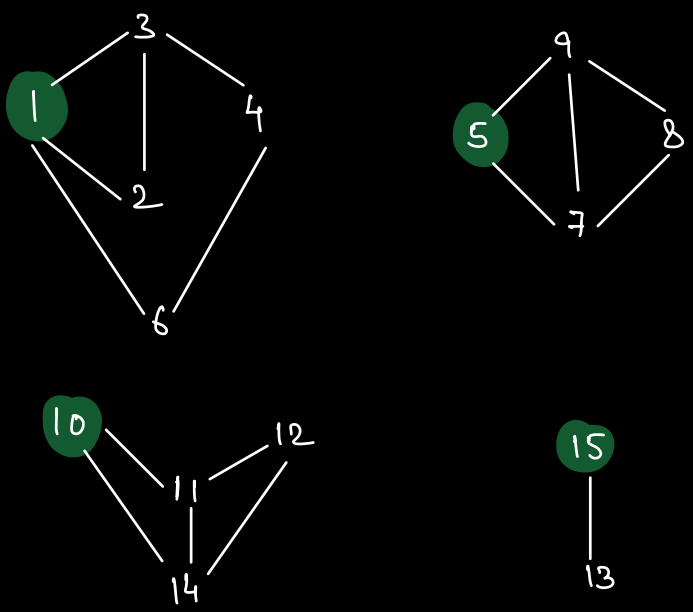
⇒ NOTE : Dfs won't always give you the shortest path b/w 2 nodes.

Only visit ⇒ Dfs

Shortest path ⇒ Bfs.

Q: Given an undirected graph, find the no. of connected components.

Connected Component: A component is said to be connected if from every node we can visit all the nodes inside the component.

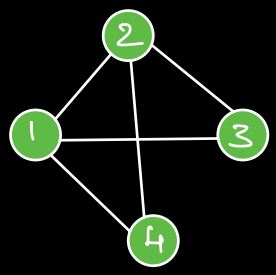


Adj. List

⇒ Inside one component if we apply BFS/DFS then all the nodes of that component will be visited.

⇒ So apply DFS/BFS algo. on all the nodes.

No. of connected components ≡ No. of BFS/DFS func^n calls.



No. of connected components = 1

No. of connected components = 6

```
int components(N, E, u[], v[]){
    //Create Adjacency list.
    list<int> g[N+1];
    bool vis[N+1] = false;
    c=0
    for(i=1; i<=N; i++){
        if(!vis[i]){
            dfs(g, N, E, i, vis);
            c++;
        }
    }
    return c;
}
```

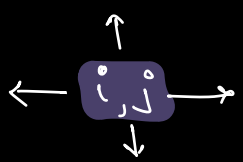
TC : O(E)

SC : O(E)

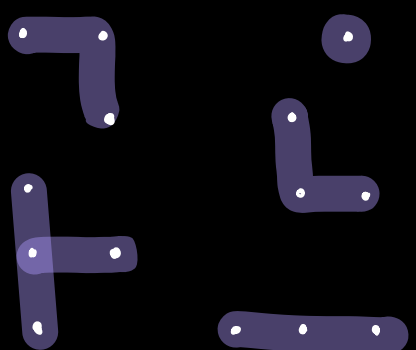
Q. **No. of Islands.** → (N×M)

Given a matrix of 1's & 0's. find the no. of islands in the matrix.

1 ⇒ land mass
0 ⇒ water body.



	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



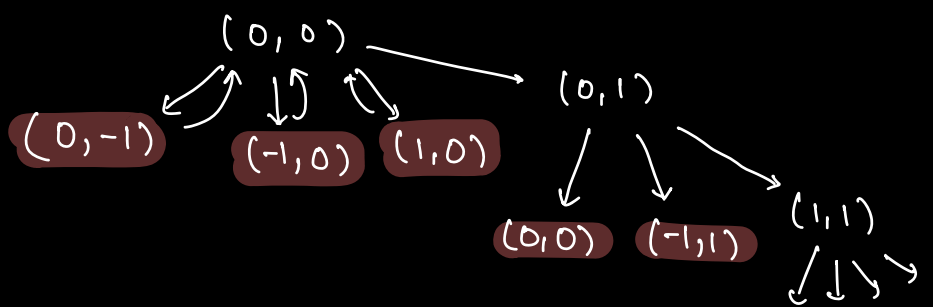
No. of islands = 5

↳ No. of connected components.

⇒ Adjacency list isn't required as we can use the given matrix itself.

	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

No. of islands ≡ No. of connected components.




```
int islands(int mat[][], N, M){
    int c = 0;
    for(i=0; i<N; i++){
        for(j=0; j<M; j++){
            if(mat[i][j] == 1){
                dfs(mat, N, M, i, j);
                c++;
            }
        }
    }
    return c;
}
```

```
void dfs(int mat[][], N, M, i, j){
    if (i<0 || j<0 || i==N || j==M || mat[i][j]==0)
        return;
    mat[i][j] = 0;
    dfs(mat, N, M, i+1, j) // Down
    dfs(mat, N, M, i-1, j) // Up
    dfs(mat, N, M, i, j+1) // Right
    dfs(mat, N, M, i, j-1) // left
}
```

i+x[0], j+y[0]

i+x[1], j+y[1]

```
int x[] : [1, -1, 0, 0]
int y[] : [0, 0, 1, -1]

for(k=0; k<4; k++){
    dfs(mat, N, M, i+x[k], j+y[k];
}
```

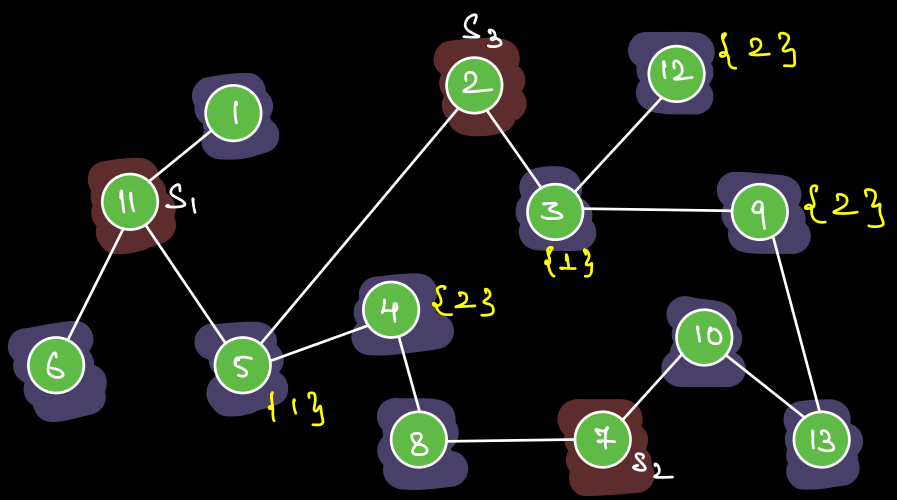
TC : O(NM)

SC : O(NM)

↳ Stack size.

Multi source BFS

Given N nodes & multiple sources S_1, S_2 & S_3. Find the length of shortest path for all the nodes to any of the source nodes.



S_1 : 11
S_2 : 7
S_3 : 2

Queue

~~2~~, ~~7~~, ~~11~~ | ~~5~~ ~~3~~ ~~8~~ 10 1 6 | 4 9, 12 13

level = 0 | level = 1 | level = 2.

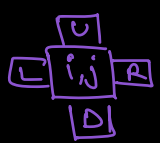
⇒ Only change is to push all the source nodes in the Queue at the start.

⇒ HW

Q. Rotten Oranges.

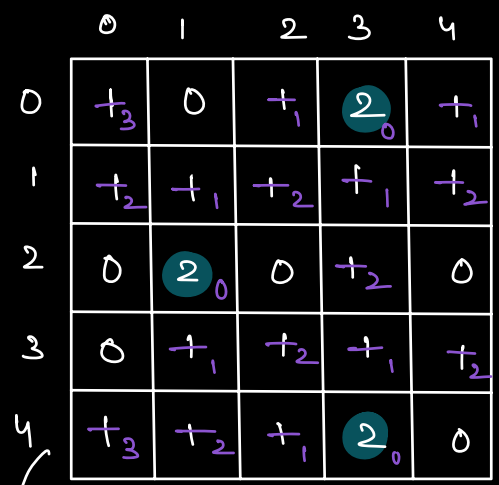
mat[N][M] →
- 0 : Empty
- 1 : fresh orange
- 2 : Rotten orange

Every minute any fresh orange, adjacent to a rotten orange will become rotten.
Find the min time in which all the oranges will become rotten.
If NOT possible, return -1;

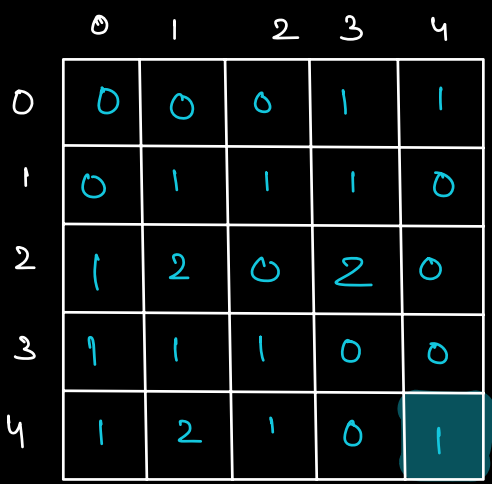


	0	1	2	3	4
0	+3	0	+3	0	+5
1	+2	1	+2	+3	+4
2	0	2	0	+4	0
3	0	1	+2	+3	+4
4	+3	+2	+3	0	+5

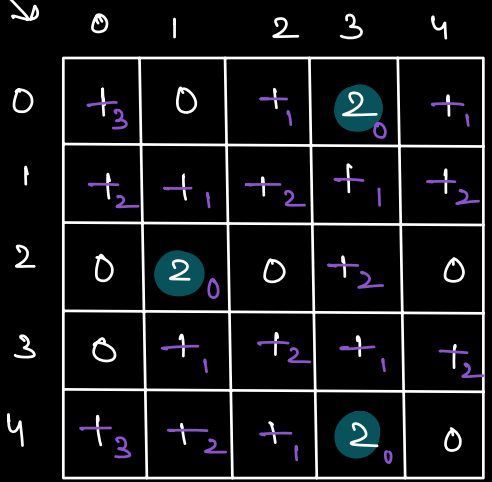
⇒ ans = 5



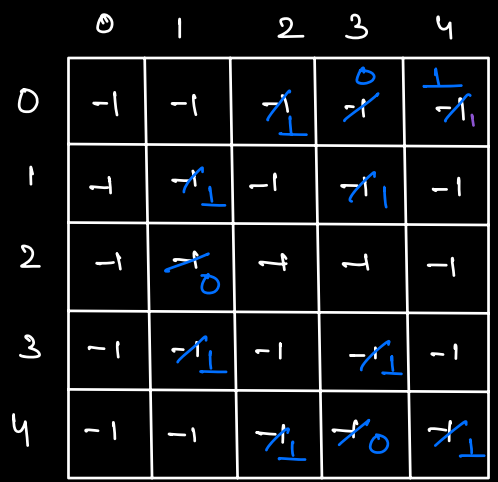
⇒ t = 3 : Ans.



⇒ -1

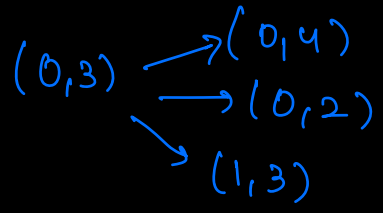


time[][] = -1



~~(0,3)~~ ~~(2,1)~~ (4,3) (0,4) (0,2) (1,3)

⇒ Queue<pair<int, int>> q;



⇒ After creating the time matrix, find max.

⇒ After the complete iteration if matrix still contains (1) then return -1.

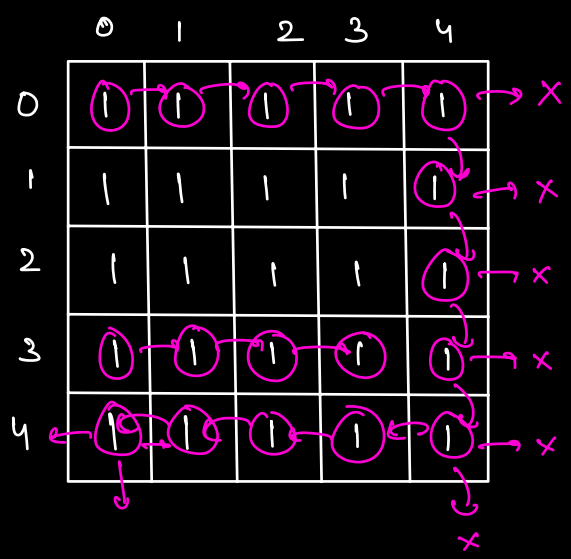
⇒ Create a separate time[][] matrix.

	0	1	2	3	4
0	t_3	0	t_3	0	t_5
1	t_2	t_1	t_2	t_3	t_4
2	0	2_0	0	t_4	0
3	0	t_1	t_2	t_3	t_4
4	t_3	t_2	t_3	0	t_5

HW : Implement

*

time[i][j] = time of source + 1 cell



R
D
L
U

O(NM)