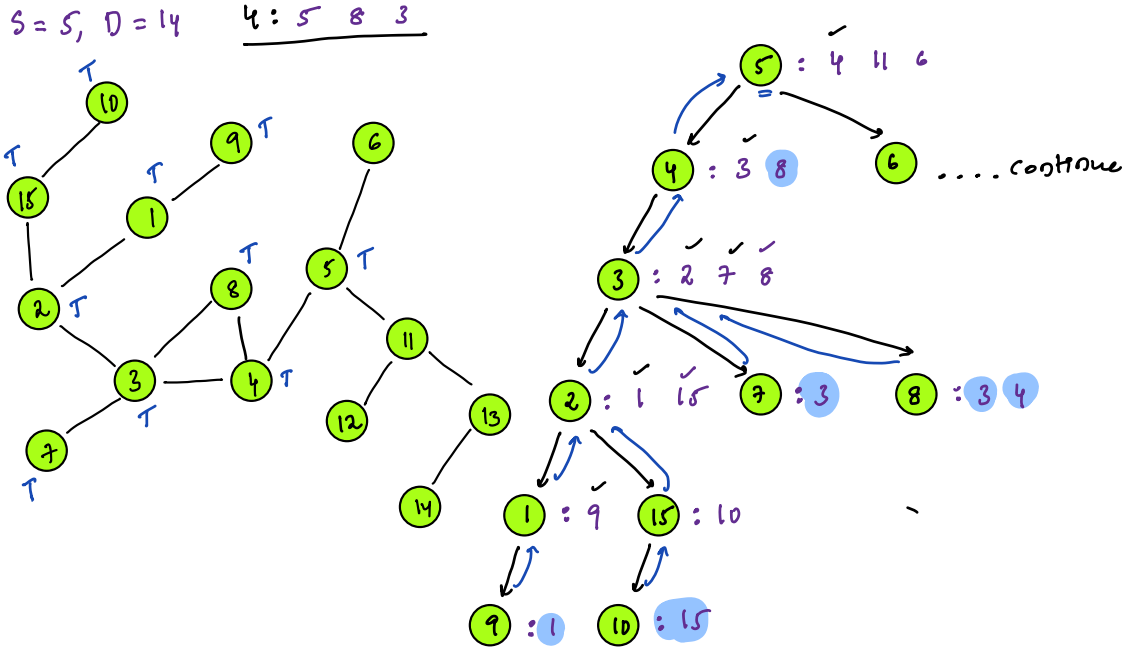
Todays Content:

- DFS ✓
- No: of connected components ✓
- No: of Islands ✓
- MultiSource BFS ✓
- Rotten Oranges ✓

DFS: Depth First Search → Similar to pre-order

S = 5, D = 14 4 : 5 8 3



Code:

Edges Information

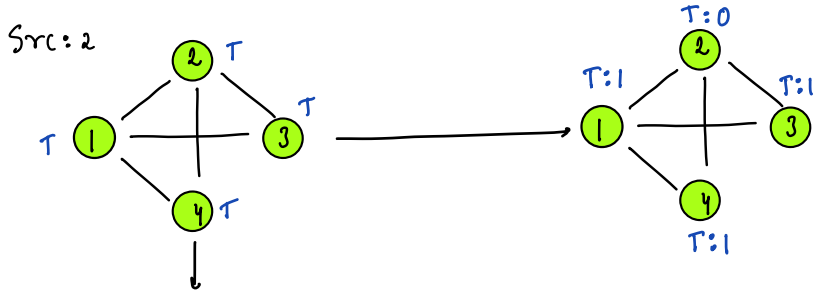
```
bool path(int N, int E, int u[], int v[], int s, int d){
    Step1: Create Adj list
           list<int> g[n+1] : TODO
    Step2: bool vis[n+1] = fala
           DFS(s, g[], vis[])
           return vis[d]
}
```

TC: O(N+E)

SC: O(N+E) = O(N) + O(N) + Stack Size : O(N) = O(N+E)

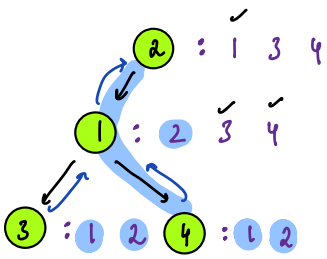
```
void DFS(int u, list<int> g[], int vis[]){
    if(vis[u] == True){return} // If we visit a node which is already visited return
    vis[u] = True // marking node as visited
    // Apply DFS to it's unvisited adj nodes
    i = 0; i < g[u].size(); i++){
        int v = g[u][i] // u ——— v
        if(vis[v] == fala){ DFS(v, g[], vis[]) }
    }
}
```

Shortest length using DFS:



x | x x x

DFS s = 2



Note: 1. Shortest path will be given BFS

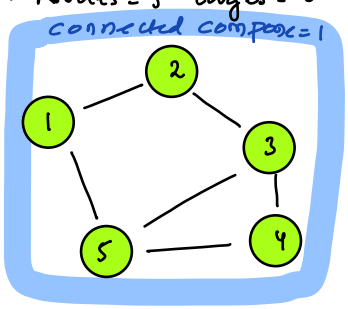
2. DFS cannot give you shortest path

Q) Checking we can visit or not?

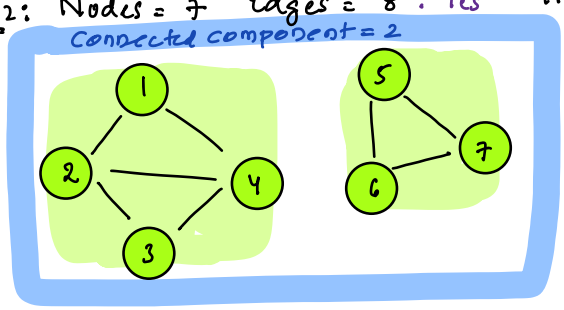
a. DFS because easy to implement

Graph Yes/No:

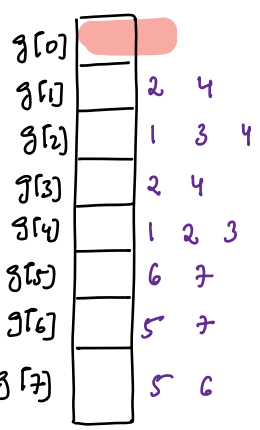
Ex1: Nodes = 5 Edges = 6 : Yes



Ex2: Nodes = 7 Edges = 8 : Yes



list<int> g[8]



28) Given **undirected Graph** find no of **Connected Components:**

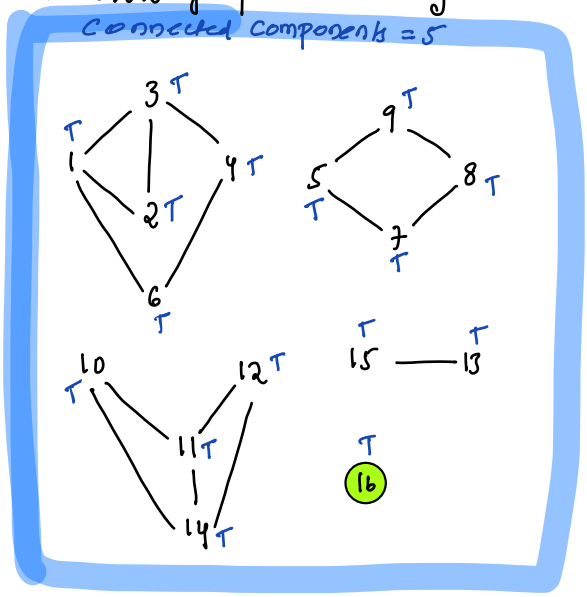
A component is said to be connected,
if From every node **we can visit all nodes** in Component

: undirected graphs

10:12 → 10:22 pm

Eg: N = 16 Indicates.

In below graph how many connected components are there?



1 single graph

N = 16, E = 16,

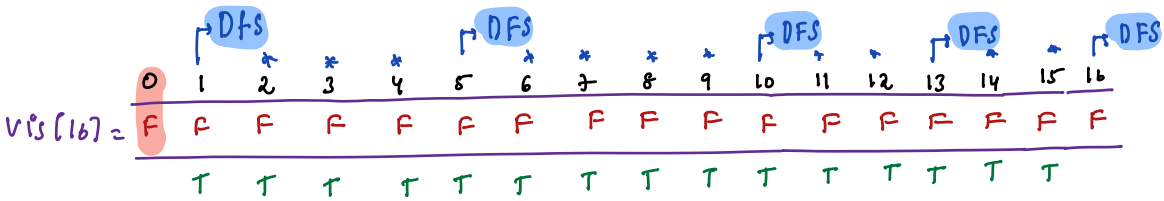
Note: A single node is also a connected component.

Input:

N = 16 E = 16

1	3	5	7	10	14
1	2	9	8	15	13
1	6	7	8	3	2
3	4	10	11	12	14
6	4	11	12		
5	9	11	14		

Idea: Count no: of times we apply BFS/DFS till all nodes visited.




```
int nCC( int N, int E, int u[], int v[])   TC: O(N+E)
                                           SC: O(N+E)
                                               O(N)
                                               O(N)   } = O(N+E)
                                           Stack Size: O(N)
  Step1: Adj list
         list<int> g[n+1] // TODO
  Step2: Connected Components
         int vis[N+1] = false
         int c = 0
         for(i = 1; i <= N; i++){
         }  if(vis[i] == false) { DFS(i, g[], v[]) c = c+1 }
         return c;
}
```

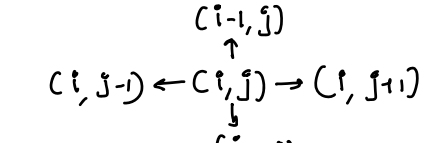
38) No: of islands ?

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

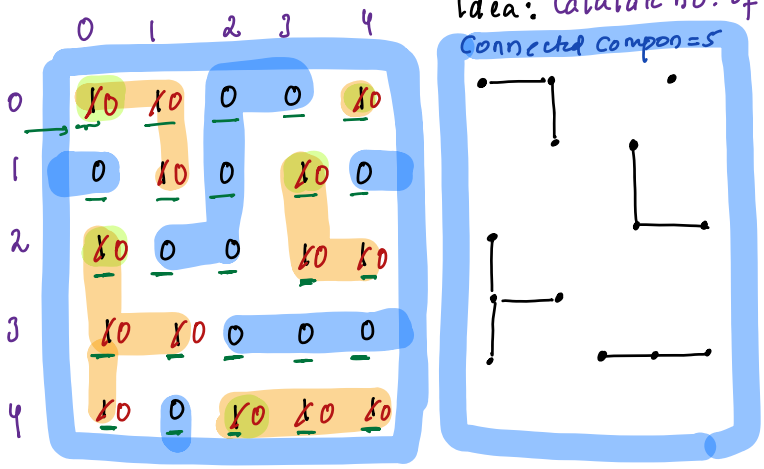
mat[][] → 1: land Island: Water in all 4 directions
↳ 0: water

Note:

Outside mat[][], assume water all in 4 directions



mat[5][5]: // 5 islands



Idea: Calculate no: of connected Components.

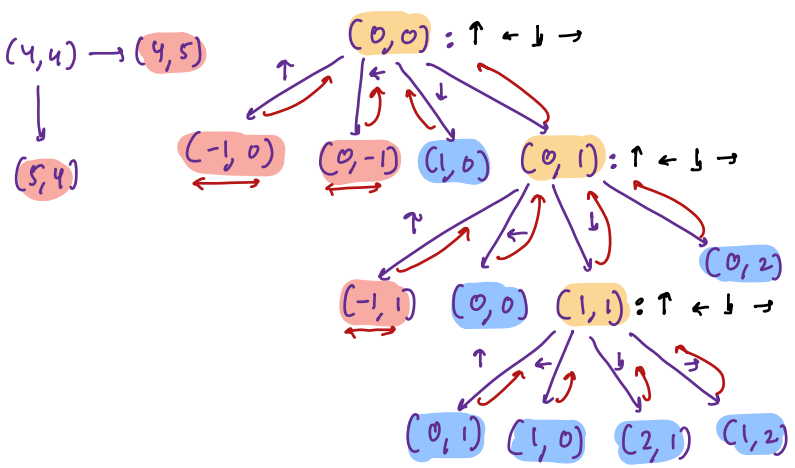
Note: 1 is indirectly repres node

Apply BFS/DFS

a) Adj list: Any given cell
We can at max visit 4 cells
Cells: (i, j)
Adjacent cells:
(i-1, j) (i+1, j) (i, j-1) (i, j+1)
Since we can get Adjacent cells inf, we dont need to create Adj list

b) Vis[]: If a node visited make that cell water so We can no longer use it

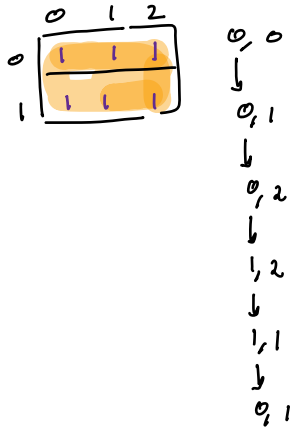
Idea: Iterate on mat[][]
: if a cell val value is 1 apply DFS
: increase count



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

TC: $4*(N*M) \approx O(N*M)$

SC: $O(N*M)$ // If all are 1's

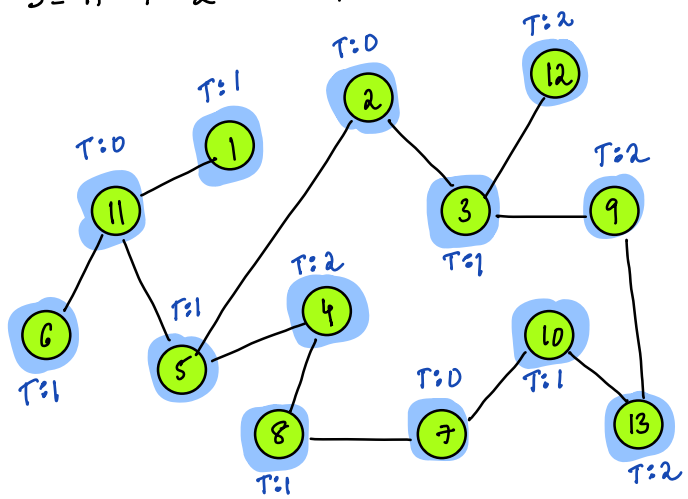



```
void dfs(int mat[N][M], int i, int j){
    if( i<0 || j<0 || i>=N || j>=M || mat[i][j]==0 ) {return}
    mat[i][j] = 0 // mark as visited
    //Call Adj Cells:
    DFS(mat, i-1, j)
    DFS(mat, i+1, j)
    DFS(mat, i, j+1)
    DFS(mat, i, j-1)
}
```

MultiSource BFS:

Given N Nodes & multiSource S_1 S_2 S_3 find length of shortest path for given dest node to any one of the source node $\{S_1, S_2, S_3\}$

Eg: S = 11 & 2 D : 4



level 0 1 2

0	1	2	3	4	5	6	7	8	9	10	11	12	13
dis:	1	0	1	2	1	1	0	1	2	1	0	2	2

Idea: At start push all source nodes at once

Apply Enact BFS

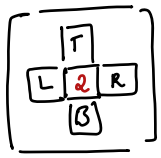
Return dist of dest node

TC: O(N+E) SC: O(N+E)

Rotten Oranges:

mat[N][M] →
- 0: Empty cell
- 1: fresh orange present
- 2: rotten orange present

Every minute any **fresh orange**, adjacent to a **rotten orange** **becomes rotten**, find min time, when all oranges become rotten. **If not possible return -1?**



Ex:

	0	1	2	3	4
0	✗5	0	✗1	2	✗1
1	✗4	✗3	✗2	✗1	✗2
2	0	✗4	0	✗2	0
3	0	✗5	✗4	✗3	✗4
4	✗7	✗6	✗5	✗4	✗5

ans: after 7 min all oranges become rotten.

Ex2:

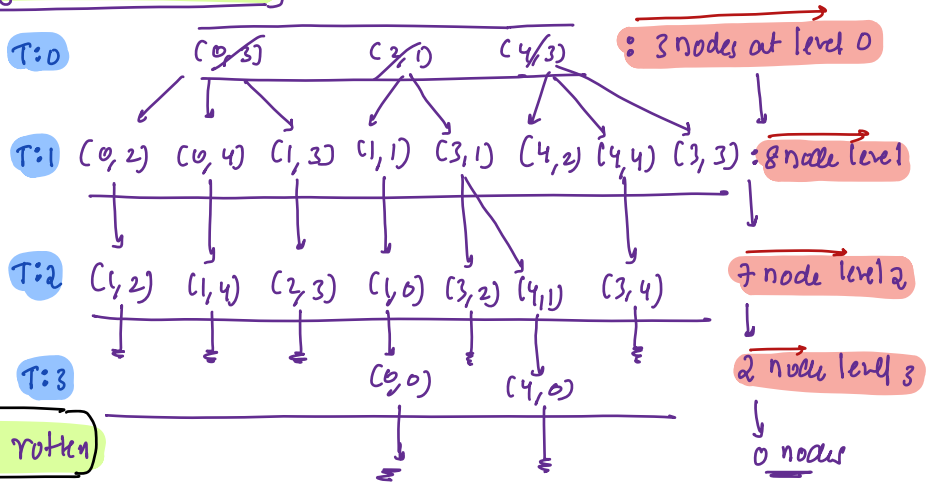
	0	1	2
0	1	0	✗2
1	0	2	✗1
2	0	✗1	0

return -1, can never become rotten.

Ex3:

	0	1	2	3	4
0	✗3	0	✗1	2	✗1
1	✗2	✓1	✗2	✗1	✗2
2	0	2	0	✗2	0
3	0	✗1	✗2	✗1	✗2
4	✗3	✓2	✗1	2	✗1

ans: after (3) min all oranges rotten



Idea: MultiSource BFS.

- Queue<pair<int,int>>: Push all rotten oranges at start itself
- No need of Adj list, because we can calculate all Adj Nodes
- Un given mat[][] itself to mark it as rotten.
- Count no: of levels -1 to get time?

```
int rottenOranges(int mat[N][M])   TC: 4*(N*M) ≈ O(N*M)
                                   SC: O(N*M)
{
    Queue<pair<int,int>> que;
    for(i=0; i<N; i++){
        for(j=0; j<M; j++){
            if(mat[i][j]==2) {q.push(pair(i,j))}
        }
    }
    int l=0 // Before pushing an orange mark rotten
    while(q.size()>0){
        int n = que.size()
        l = l+1                          // We delete all nodes at a level
        for(i=0; i<n; i++){              // Calculating no: of level are there =   ans = level-1
            pair<int,int> ele = q.front()
            q.delete() // delete front element
            int x = ele.first, int y = ele.second
            // Adj nodes of x,y = (x-1,y) (x+1,y) (x,y-1) (x,y+1)
            if(x-1>=0 && mat[x-1][y]==1){ mat[x-1][y]=2 que.push(pair(x-1,y))}
            if(x+1<n  && mat[x+1][y]==1){ mat[x+1][y]=2 que.push(pair(x+1,y))}
            if(y-1>=0 && mat[x][y-1]==1){ mat[x][y-1]=2 que.push(pair(x,y-1))}
            if(y+1<M  && mat[x][y+1]==1){ mat[x][y+1]=2 que.push(pair(x,y+1))}
        }
    }
    for(i=0; i<N; i++){
        for(j=0; j<M; j++){
            if(mat[i][j]==1){return -1}
        }
    }
    return l-1
}
```