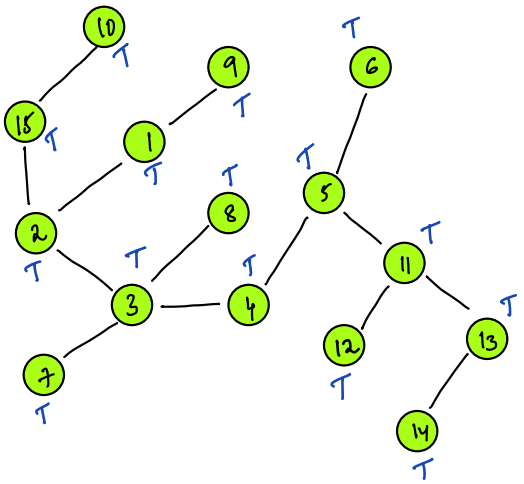
Todays Content :

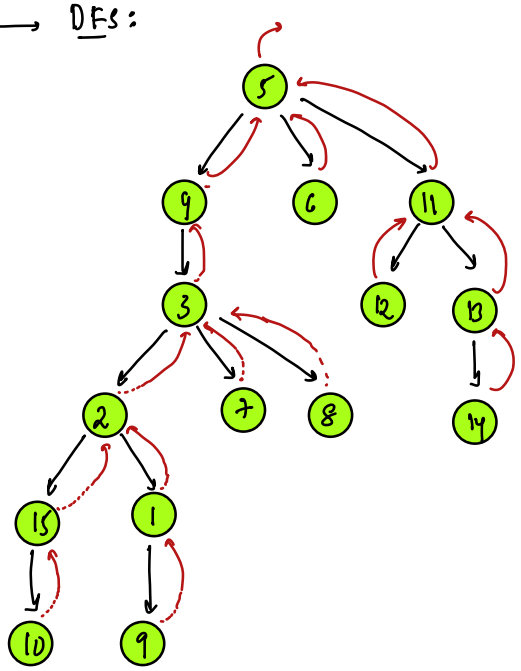
- dfs Content
- No of Connected Components
- # No of islands
- Multi Source BFS
- Rotten Oranges
- Graph Colouring

DFS Content: S = 5, D = 14 → return True

↳ Depth First Search



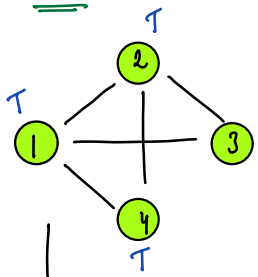

→ DFS:



Pseudocode: TC → O(E)  SC → O(E + N + N) → O(E)  (stack space)

```
bool path(int N, int E, int u[], int v[], int s, int d){
    list<int> g[N+1] // TODO
    bool vis[N+1] = {F}
    dfs(g, vis, s)  // recursion code, fill vis[]
    return vis[d]
}
void dfs(list<int> g[], bool vis[], int s)
    if(vis[s] == True) {return} // if node already visited directly return
    vis[s] = True
    for(int i=0; i<g[s].size(); i++){
        int v = g[s][i] -> // irrespective whether v is visited let's call the function
    }
}
```

// Eg: Src → 2

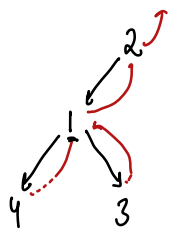



BFS:

| 0 | 1 |
|---|---|
| ~~2~~ | ~~1~~ ~~3~~ ~~4~~ |

↳ from 2 shortest distance to reach 1, 3, 4 = 1

↳ DFS:




Length of path from 2 → 4 : 2

Obs: DFS won't give you length of shortest path from S → D

Note: Only Visition → DFS, Shortest Path → BFS

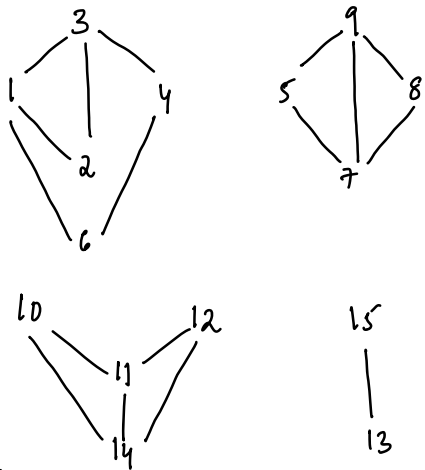
20) Given **undirected graph** find no: of connected Components

A component is said to be connected, if from every node we can visit all nodes inside component

Ex: N = 15 → Indicates 15 Nodes

In below graph how many connected components are there

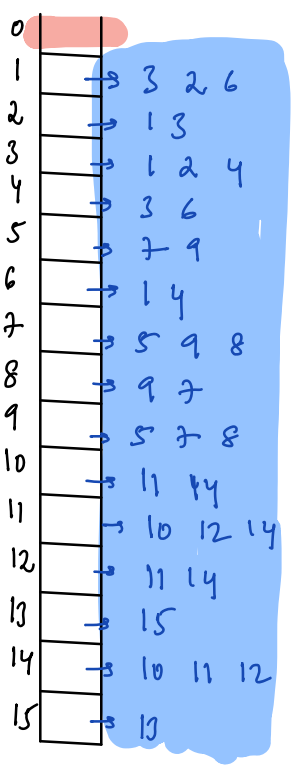
ans: 4



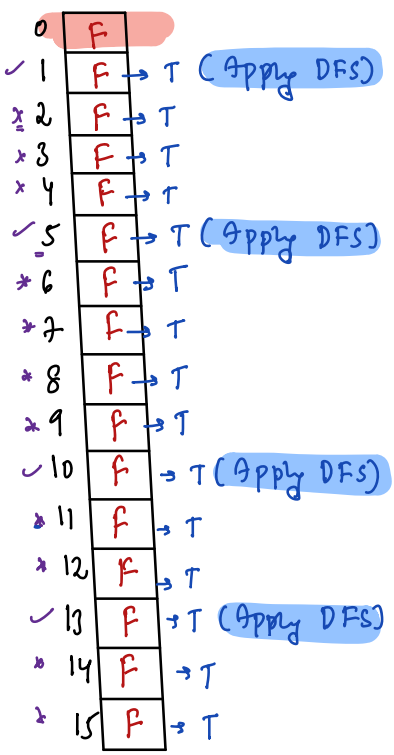

Idea: Inside connected component if we apply BFS/DFS with any node, all nodes inside component will be visited

# No: of time we apply DFS till all nodes are visited = Connect Components

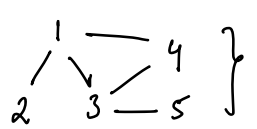
Adj list:




vis [16]




Note:




# connected component = 1

2: 1 2 3 4 5 } # Connect components = 5

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

```
int components(int N, int E, int u[], int v[]){
    list<int> g[N+1] // TODO
    bool vis[N+1] = {F}
    int c=0
    for(int i=1; i<=N; i++){
        if(vis[i]==false){
            dfs(g, vis, i) // recursion code, fill vis[]
            c=c+1
        }
    }
    return c;
}

void dfs(list<int> g[], bool vis[], int s)
    if(vis[s]==true){return} // if node already visited
                            directly return
    vis[s] = true
    for(int i=0; i<g[s].size(); i++){
        int v= g[s][i] → // irrespective wether v is visited
        dfs(g, vis, v)      let's call the function
    }
}
```

## 308) No of Islands:

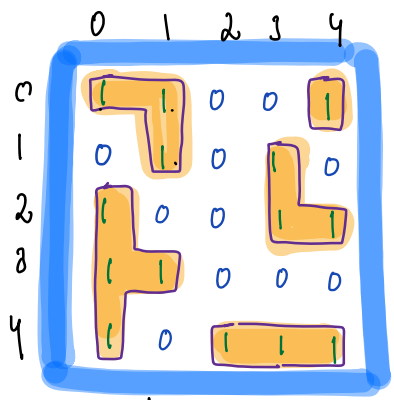
Given a matrix of 1's & 0's find no of Islands are present?

↳ If surrounded by water in all 4 directions, it's Island

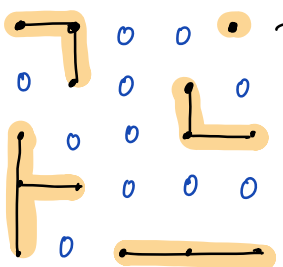
mat[][] → 1: land, 0: water

mat[5][5] → // Given Matrix

(i-1, j)
↑
(i, j-1) ← (i, j) → (i, j+1)
↓
(i+1, j)

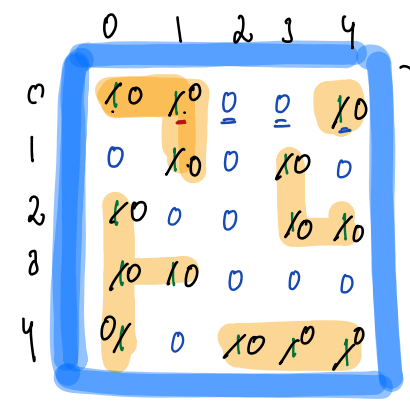


// Islands ⇒ 5



# no: of connected components

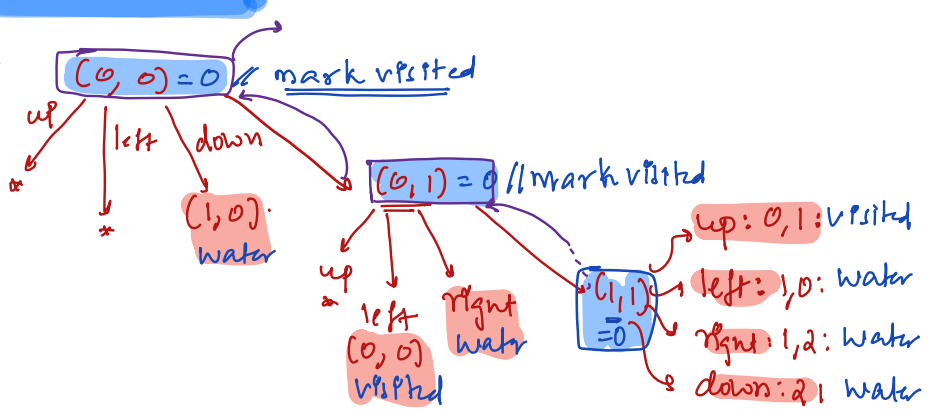
Obs1: Since for every cell we already known all 8 possible steps we don't need adj list



Total we applied dfs 5 times
↳ # 5 lands

Use given matrix itself to mark visited & unvisited

→ Trace:

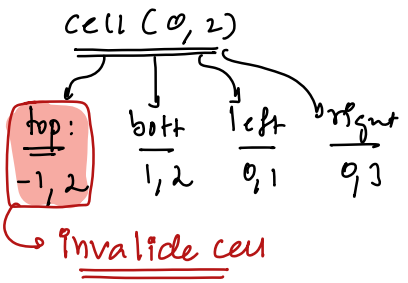


Pseudocode : TC: O(N*M) SC: O(1 + N*M) → Recursive stack)

```
int islands(int mat[][], int N, int M){
    int c=0
    for(int i=0; i<N; i++){
        for(int j=0; j<M; j++){
            if(mat[i,j]==1){
                // with cell i, j as source apply DFS
                c++;
                dfs(mat, i, j, N, M)
            }
        }
    }
    return c;
}
```

10:50 → 10:57pm

```
          T  B  L  R
int x[] = {-1 +1  0  0}
int y[] = { 0  0 -1 +1}
void dfs(int mat[][], int i, int j, int N, int M){
    if(i<0 || j<0 || i>=N || j>=M || mat[i,j]==0){return;}
    mat[i,j]=0 // make it visited
    dfs(mat, i+1, j, N, M)
    dfs(mat, i-1, j, N, M)
    dfs(mat, i, j+1, N, M)
    dfs(mat, i, j-1, N, M)
}
```



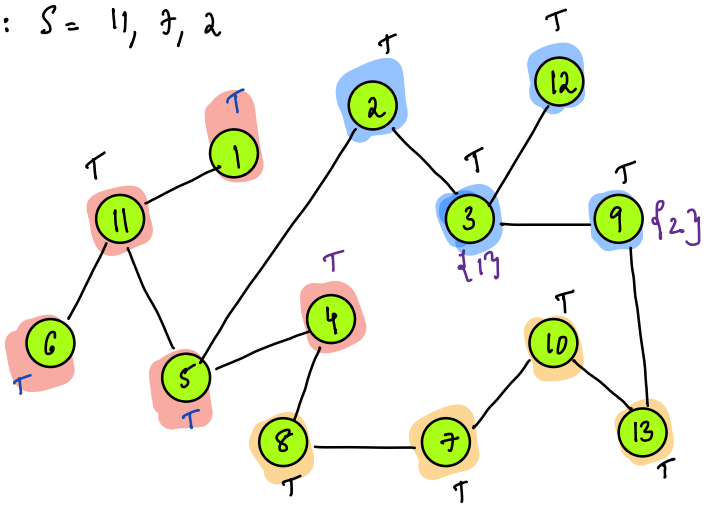

same

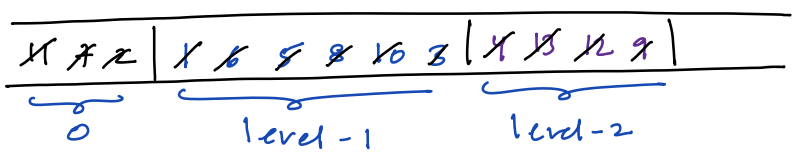
```
for(int k=0; k<4; k++){
    dfs(mat, i+x[k], j+y[k], N, M)
}
```

# MultiSource BFS

Given N Nodes & multiple Source $S_1$ & $S_2$ & $S_3$ find length of shortest path for all node to any of the source nodes $S_1$ $S_2$ $S_3$

Eg: S = 11, 7, 2



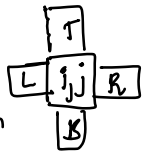




Idea: Only change is push all source nodes, initially in queue & apply bfs

↳ TODO

# Rotten Oranges

mat[N][m] → 0 empty
→ 1 fresh orange present
→ 2 rotten orange present



cells

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

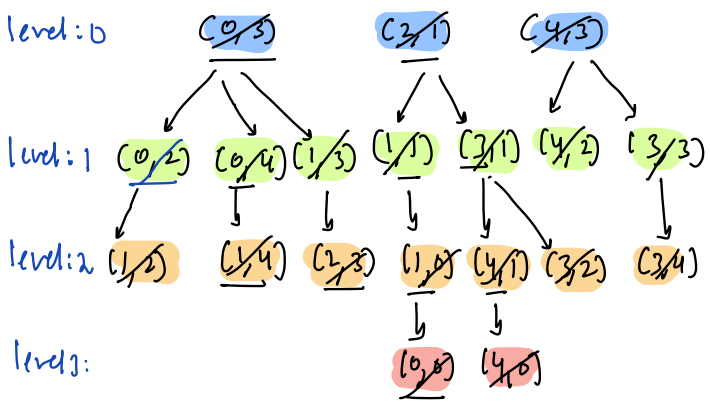
Ex1:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | ✗3 | 0 | ✗3 | 0 | ✗5 |
| 1 | 1 2 | ✗1 | 1 2 | ✗3 | ✗4 |
| 2 | 0 | 2 | 0 | ✗4 | 0 |
| 3 | 0 | ✗1 | ✗2 | ✗3 | ✗4 |
| 4 | ✗3 | ✗2 | ✗3 | 0 | ✗5 |

→ min time for all range to go bad = 5 mins, Idea = BFS

Ex2: → multisource bfs

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | ✗ | 0 | ✗ | 2 | ✗ |
| 1 | ✗ | ✗ | ✗ | ✗ | ✗ |
| 2 | 0 | 2 | 0 | ✗ | 0 |
| 3 | 0 | ✗ | ✗ | ✗ | 1 |
| 4 | ✗ | ✗ | ✗ | 2 | 0 |




→ min time to make all oranges rotten = 3

→ In mat[][] no more fresh oranges are there

```
int minTime(int mat[][], int N, int M) {        TC: O(N*M)
    Queue<pair<int,int>> q;                     SC: O(N*M)
    int time[N][M] = -1;
    // All rotten oranges inside we push in q
    for(i=0; i<N; i++){                         // Step 1
        for(j=0; j<M; j++){
            if(mat[i][j]==2){
                q.insert({i,j})
                time[i,j] = 0
            }
        }
    }
                    T   B   L   R
    int x[] = {-1   +1   0   0}
    int y[] = {0    0   -1   +1}
    while(q.size()>0){
        pair<int,int> d = q.front(); q.pop()
        int i = d.first, j = d.second
        for(k=0; k<4; k++){
            a = i+x[k], b = j+y[k]              // cell a,b
            if(a>=0 && a<N && b>=0 && b<M && mat[a,b]==1){
                mat[a,b] = 2                    // make it rotten
                time[a,b] = time[i,j]+1
                q.push({a,b})
            }
        }
    }
```

Ch1: Iterate in mat[][], even if single 1 present: return -1

Ch2: Iterate in time[][], return max of time[][]