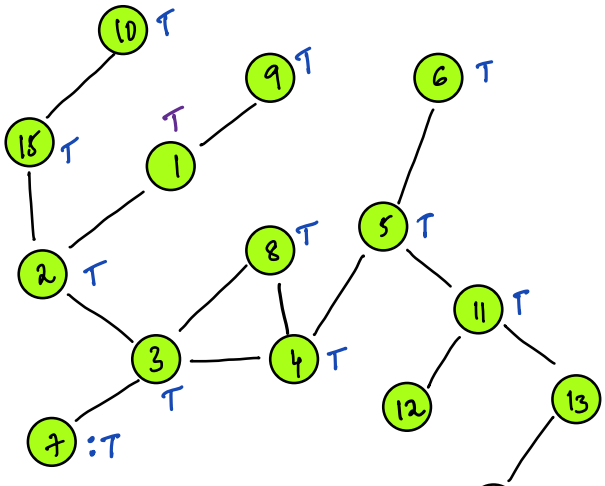
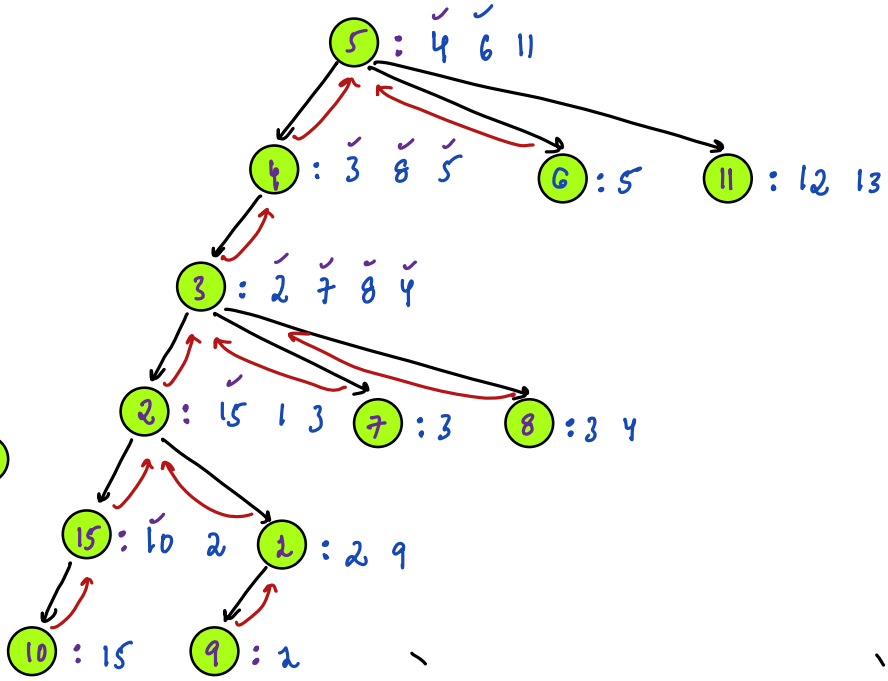
Todays Content:

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

DFS: Depth First Search → Similar to pre-order

S = 5, D = 14



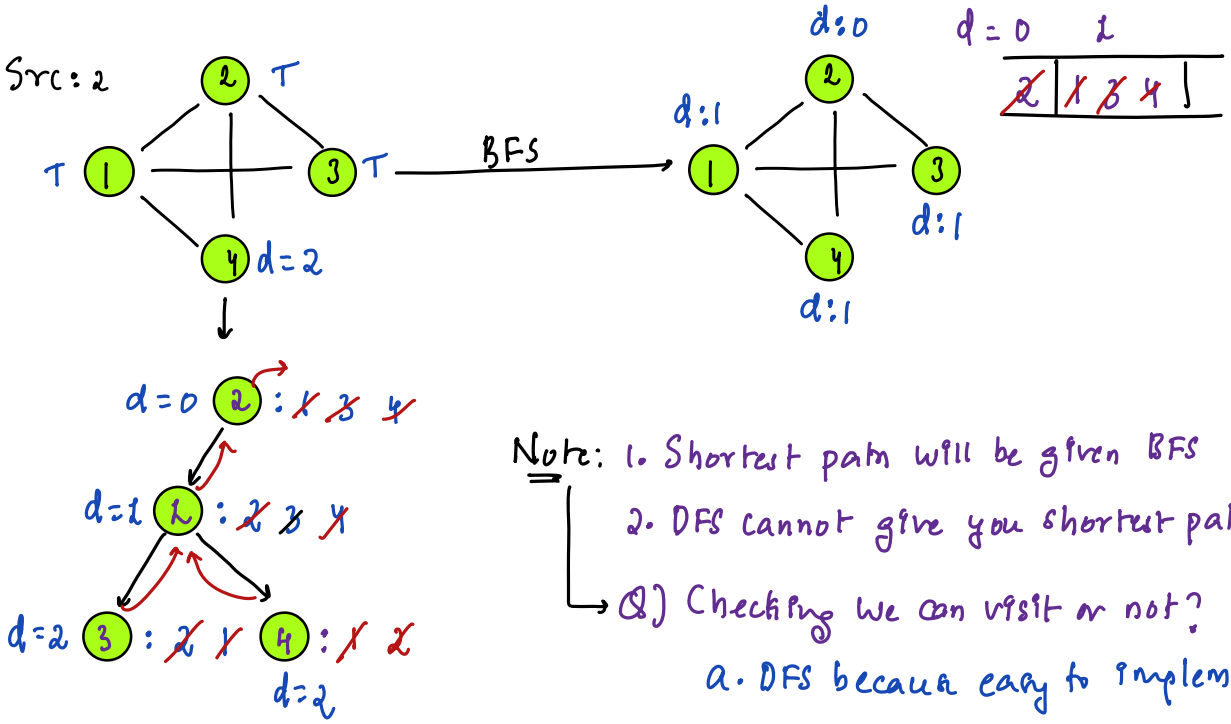


Code:

Edges Information → source → dest

```
bool path(int N, int E, int u[], int v[], int s, int d){   TC: O(N+E)
                                                           SC: O(N+E)
    Step1: Create Adj list
        ArrayList<ArrayList<Integer>> g = new ArrayList<>();
        // TODO
    Step2: bool vis[N+1] = false;
        DFS(s, g, vis); // Calling DFS traversal from source
        return vis[d]; // If we can return destination or not.
}
void DFS( int u, ArrayList<ArrayList<Integer>> g, bool vis[]){
    if( vis[u] == True) { return }
    vis[u] = True;
    // cal dfs for adjacent nodes of u.
    for(int i=0; i<g.get(u).size(); i++){
        int v= g.get(u).get(i);
    }   DFS(v, g, vis)
}   return;
```

Shortest length using DFS:



Note: 1. Shortest path will be given BFS

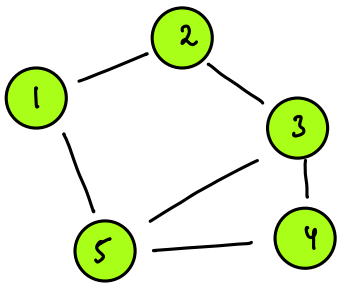
2. DFS cannot give you shortest path

→ Q) Checking we can visit or not?

a. DFS because easy to implement

Graph Yes/No:

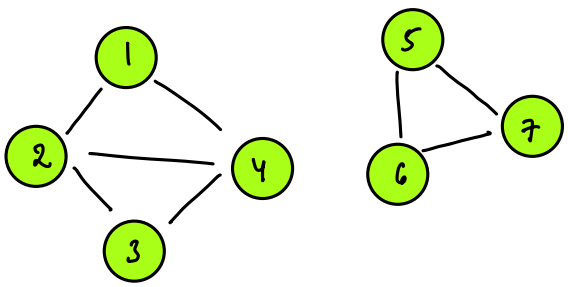
Ex1:



Nodes = 5 Edges = 6 :

Components = 1

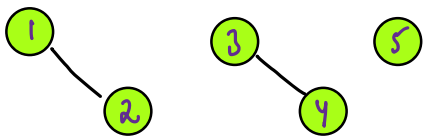
Ex2:



Nodes = 7 Edges = 8 :

Components = 2

Ex3:



Nodes = 5 Edges = 2 :

Components = 3

28) Given undirected Graph find no of Components:

If From every node we can visit all nodes in Component

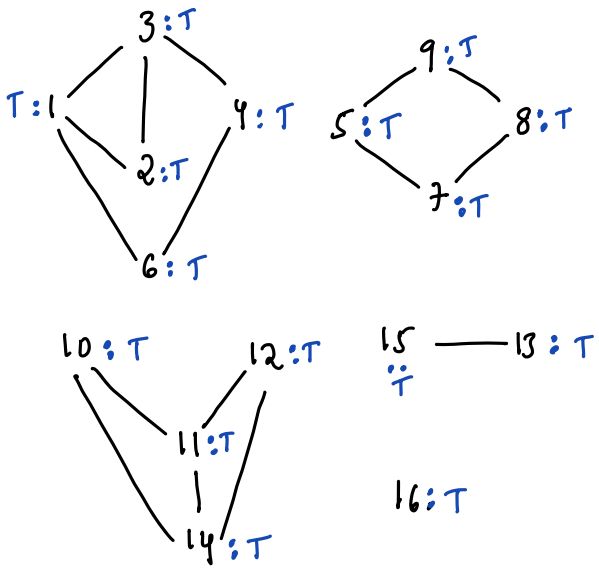
Ex: N = 16 indicates.

In below graph how many connected components are there?

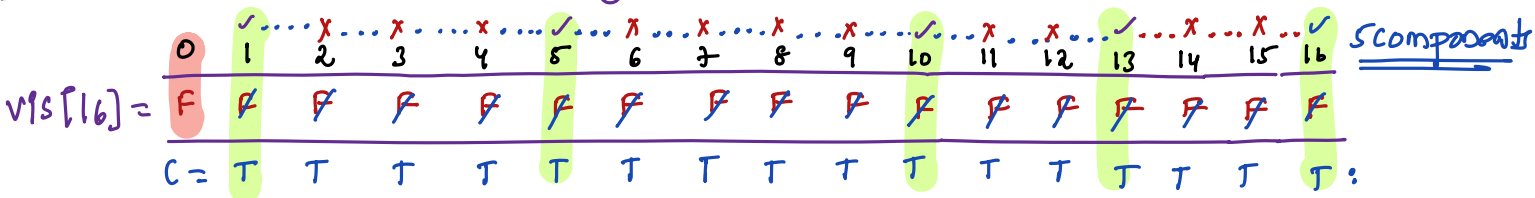
Connected Components =

1 single graph

N = 16, E = 16



Idea: Iterate on all nodes, & apply DFS on unvisited & inc count.



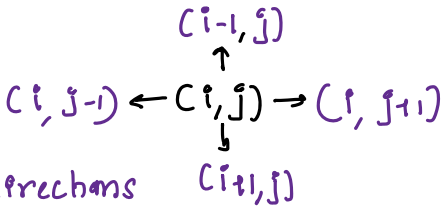

```
int nCC( int N, int E, int u[], int v[])   TC: O(N+E)
                                           SC: O(N+E)
  Step1: Adj list
      ArrayList< ArrayList< Integer >> g = new ArrayList<>();
      TODO:

  Step2: Connected Components
      int vis[N+1] = false; c=0;
      for(int i=1; i<=N; i++){
        if( vis[i] == false) { c=c+1; dfs(i, g, vis)}
      }
      return c;
}
```

38) No: of islands ?

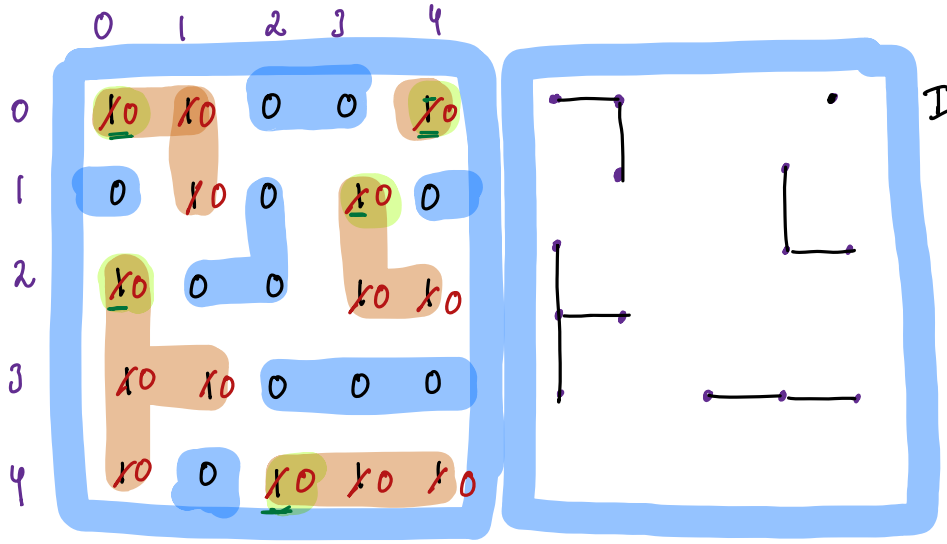
Given a matrix of 1's & 0's find no: of Islands are present?

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



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

mat[5][5]: Islands: 5



Idea: Calculate no: of components

Using DFS:

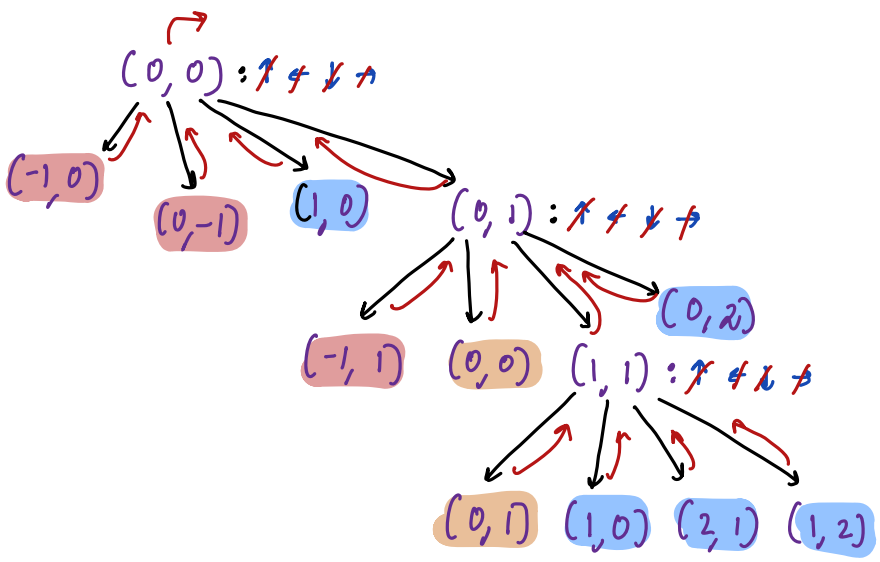
1. Adj list:

Why Adj list created?

For a given node, we want to get its immediate connected nodes

Idea: Iterate on matrix,

if a cell = 1, apply DFS & inc c++



In above question:

Cell (i, j) Adjacent cells?

(i-1, j) (i+1, j) (i, j-1) (i, j+1)

Because we can calculate on the fly no need to store.

b) Vis[]:

mat[][] = 1: not visited

mat[][] = 0: visited.

Final: Iterate on matrix, if a cell is not visited apply dfs & total count = no: of components.

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

TC: O(N*M)

SC: O(max stack size = (N*M))

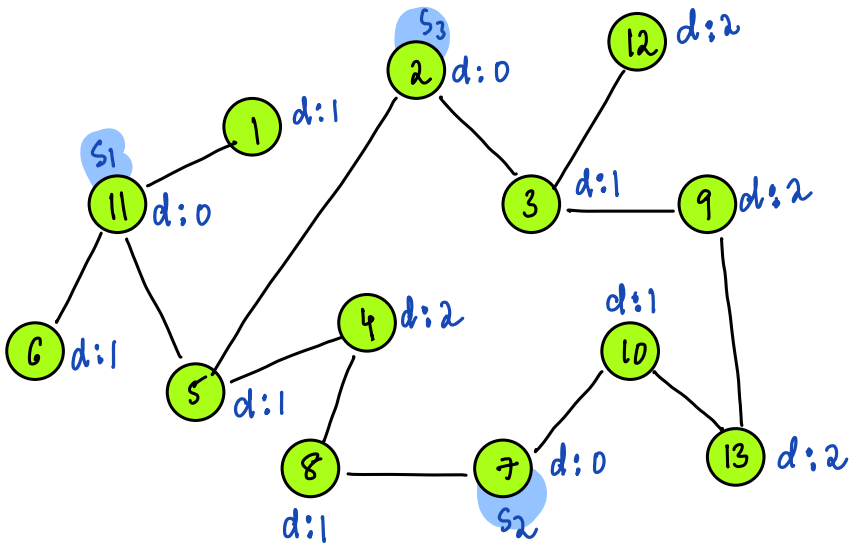
```
void dfs(int mat[][], int i, int j){
    if( i<0 || j<0 || i>=N || j>=M || mat[i][j]==0){return}
    mat[i][j]=0 // mark visited.
    // Apply DFS to adjacent nodes
    dfs(mat, i-1, j)
    dfs(mat, i+1, j)
    dfs(mat, i, j-1)
    dfs(mat, i, j+1)
}
```

(Adjacent nodes of i, j)

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 $\{S_1, S_2, S_3\}$

Ex: S = 11 7 2 D : 9 ans = { 11-9 : 4 7-9 : 3 2-9 : 2 } Final ans = 2.



level : 0 1 2

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

Q: When we want to get length of shortest path, to any one of the source node, we apply multi source BFS.

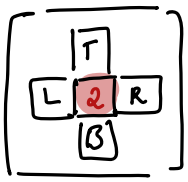
Idea: 1. Push all soura nodes in queue
2. Remaining steps are exactly same BFS.

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 time, when all oranges become rotten. If not possible return -1?



Ex1:

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

ans: after 4 min all oranges become rotten.

Ex2:

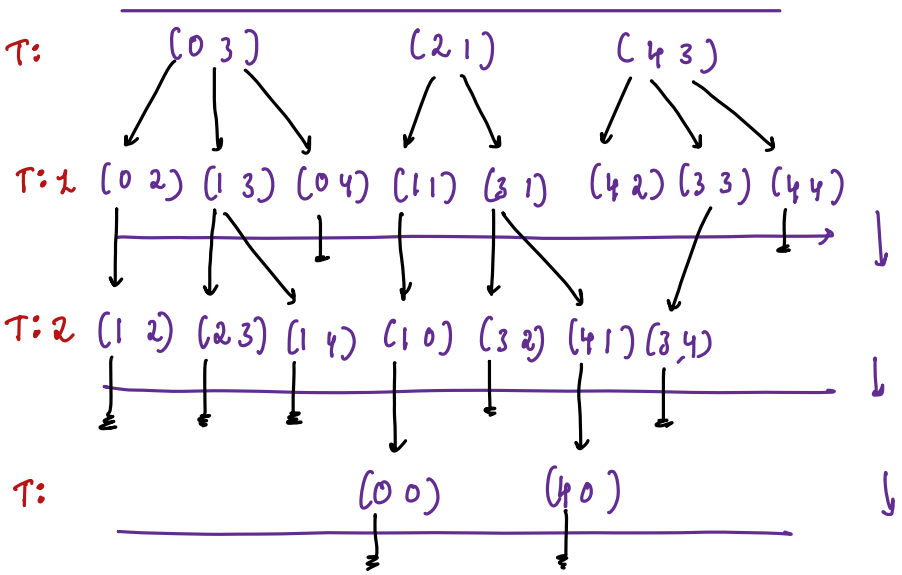
	0	1	2
0	1	0	~~1~~
1	0	2	~~1~~
2	0	~~1~~	0

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	~~0~~	~~2~~	~~1~~	2	~~1~~

After: min all rotten: 3



Idea: Multi Source BFS

1. Queue: each ele is cell (i,j), → store class Object
2. Adj list: For a given cell, we can get adj nodes: No need of list
3. Vist[]: Orange = 2: Rotten, visited Orange = 1: Unvisited.
4. Final ans: Time = No. of levels in BFS, keep a track using count.

   Iterate on entire mat[][], if a single fresh exist = return -1, else return c;

```
int rottenOranges(int mat[N][M]) // TODO
{
    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   ans = level-1
        for(i=0; i<n; i++){              // Calculating no: of level are there =
            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
}
```