Graph 1 : Introduction, DFS & Cycle Detection

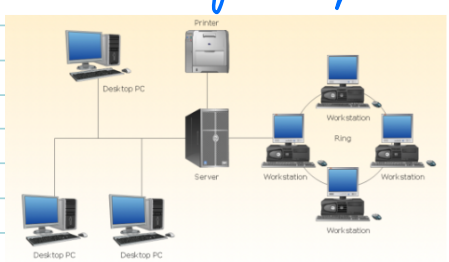
Introduction To Graphs

GRAPH :- It is simply a collection of Nodes connected to Each other using Edges.

⇒ Real life use cases

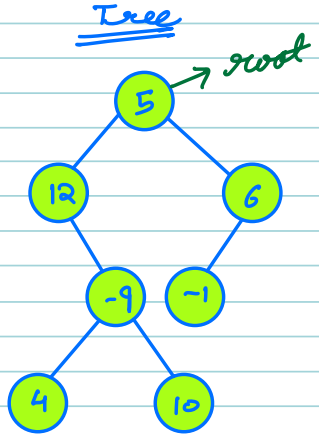
Ex1 Network of Rodes

Ex2 Network of Computers



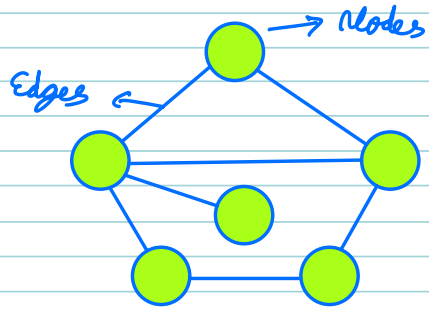
Difference b/w Trees & Graph

Tree



n = 7
E = 6

Graph

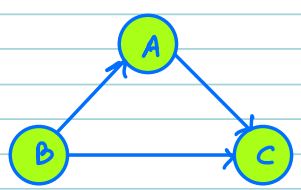


n = 6
E = 7

* In Graph we can start from any Node while in trees there is a root to start.
* Tree is a Hierarchical Data Structure
* In a Tree, No. of Edges {N-1}

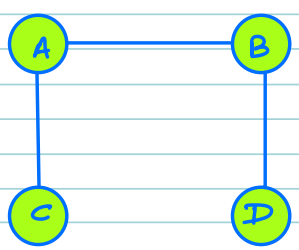
CLASSIFICATION OF GRAPHS

Case I



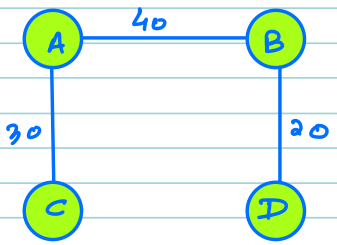
A Graph with directiond edge is called

DIRECTED GRAPH

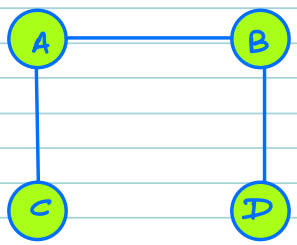


⇒ UNDIRECTED GRAPH

Case II

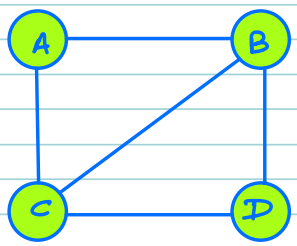


=> WEIGHTED GRAPH

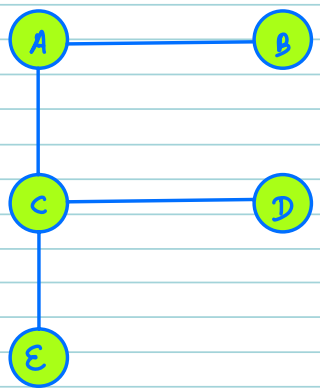


=> UNWEIGHTED GRAPH

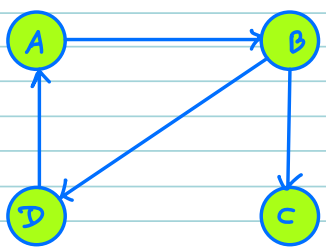
Case III



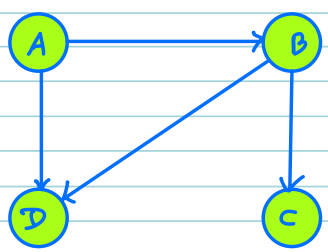
Undirected Cyclic Graph



Undirected Acyclic graph



Directed Cyclic graph



Directed Acyclic graph.

How Graph is given as Input?

↳ Any Graph is Collection of Nodes & Edges.

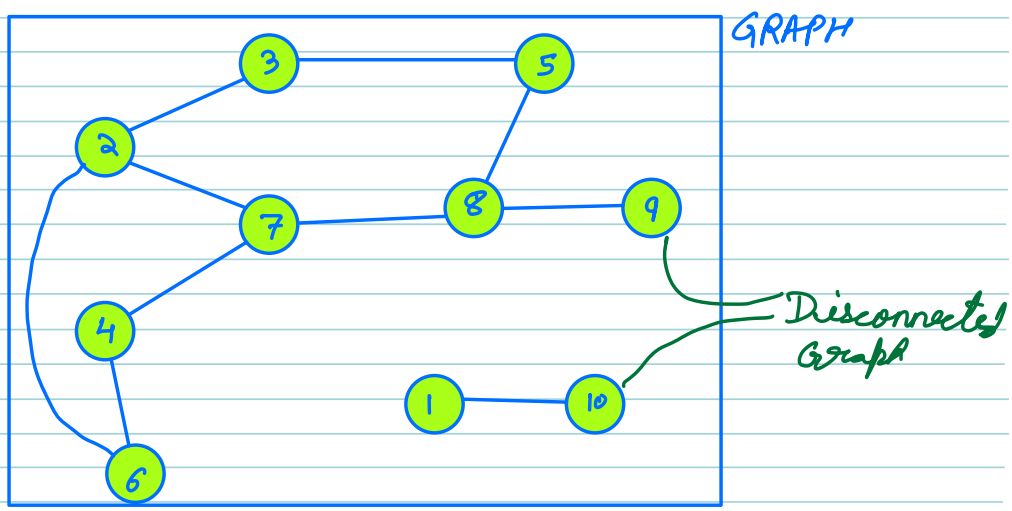
Q Given a Undirected Graph with N Nodes & E Edges.

// Input

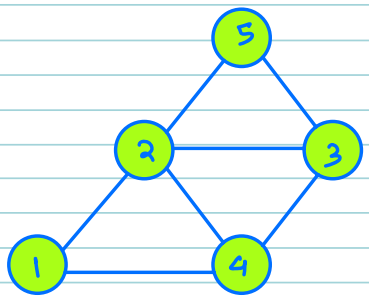
→ 1st Line N & E
→ Followed by E lines Each line contains u & v indicating edge b/w u & v nodes

N	E
10	10

u	v
2	3
4	7
8	9
2	7
7	8
10	1
4	6
5	8
2	6
3	5



STORING A GRAPH



INPUT :-

Nodes Count	Edges Count
5	7
1	4
2	5
3	2
4	3
2	4
3	5
1	2

APPROACH 1 :- Adjacency Matrix

```
int[][] mat = new int[6][6];
```

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

For edge i,j we need to mark mat[i][j] = 1 & mat[j][i] = 1 because of undirected graph

⇒ Given N & E, mat[N+1][N+1]

// u

	Unweighted	Weighted
Undirected	mat[u][v] = 1 mat[v][u] = 1	mat[u][v] = weight mat[v][u] = weight
directed	mat[u][v] = 1	mat[u][v] = weight

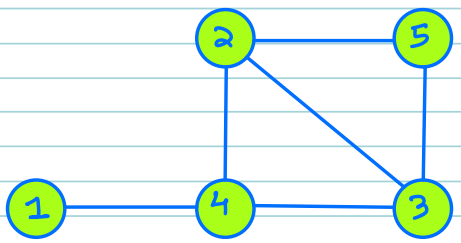
SC → O(N²)

Q. Any problem in Storing Graph using Adjacency Matrix ?

↳ Space Wastage

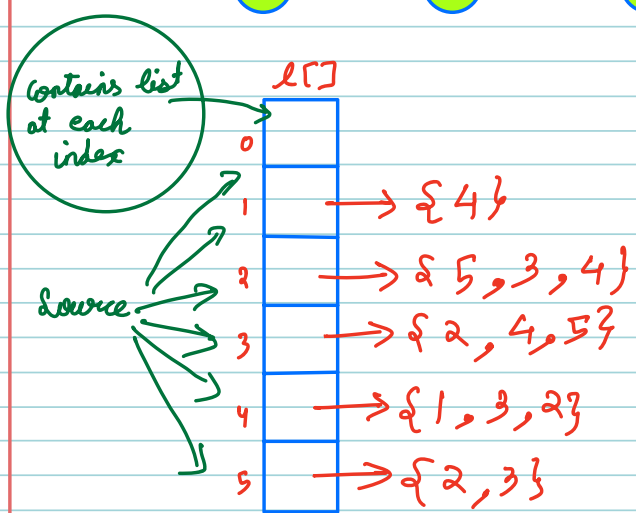
APPROACH :- Adjacency list

`List<int>[] arr = new List<int>[N+1]`



Input :-

N	E
5	6
1	4
2	5
3	2
4	3
2	4
3	5



	unweighted	weighted
undirected	l[u].add(v) l[v].add(u)	l[u].add({v, w}) l[v].add({u, w})
directed	l[u].add(v)	l[u].add({v, w})

Quiz :- Consider a graph contains V vertices & E edges. what is the SC of adjacency list

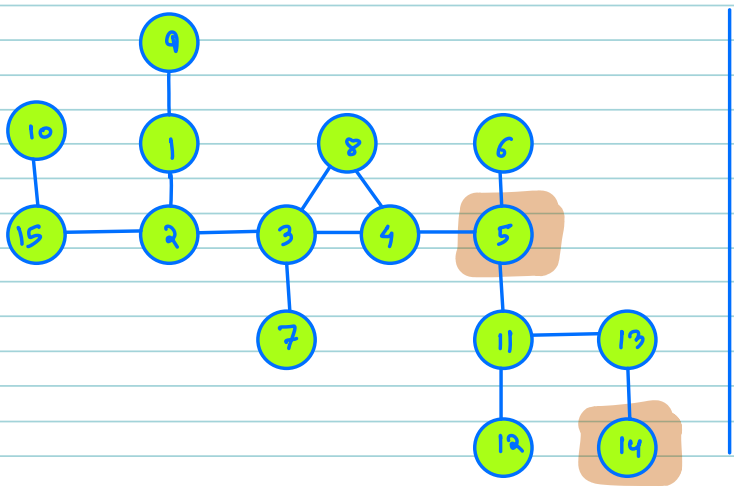
SC → O(V + 2E)
≒ O(V + E)

TRAVERSAL OVER GRAPH

⇒ DEPTH FIRST SEARCH

Source = 5, Destination = 14

Need to tell whether their is path from source to Destination



5 {6, 4, 11}

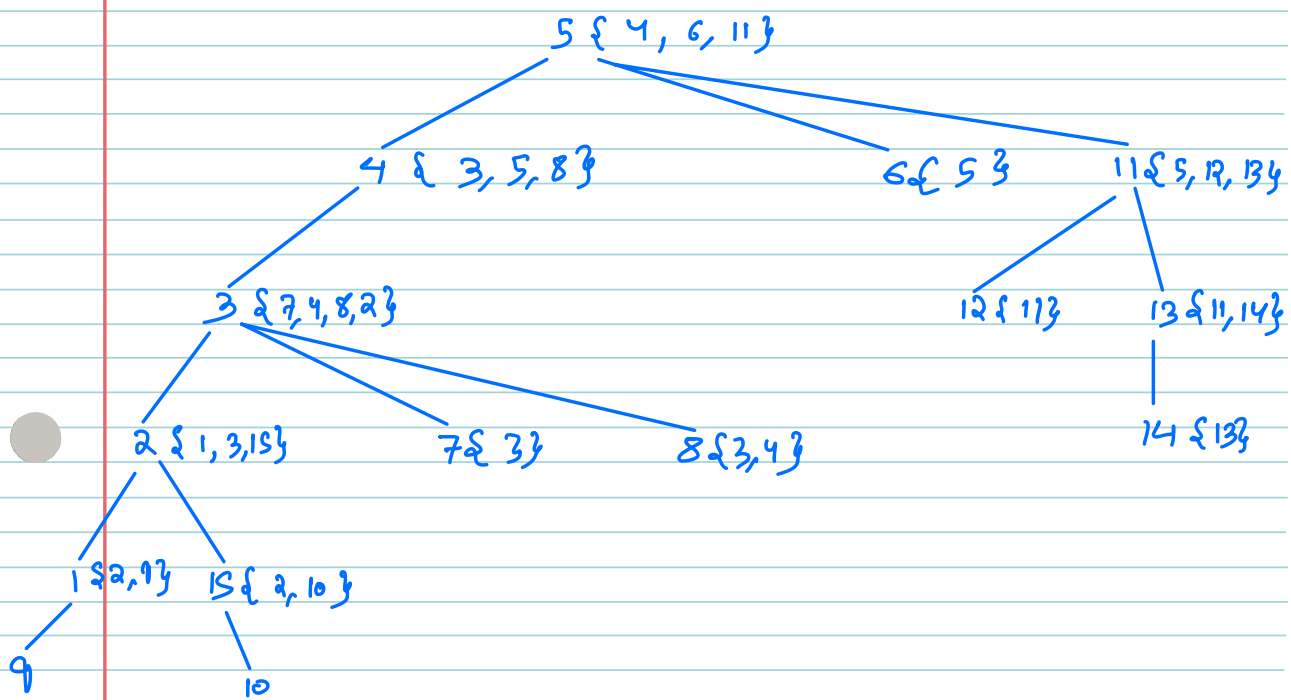
4 {5, 8, 3}

5 ← This a problem. So we need to maintain Visited Nodes.

DRY RUN

visited[16]:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T



PSEUDO CODE

```
bool Path ( int N, int E, int[] u, int[] v, int S,
                                                int D){
    ① Create Adjacency list //TODO
    ② bool vis [N+1] = {F};
                          → Adjacency list
       DFS ( S, g[], vis)
       return vis [d];
}

void DFS ( int u , list<int>[] g , bool[] vis){
    vis [u] = true;
    list < int > neigh = g [u];
```

```
        for( int i : neigh ){
            if( vis[i] == false ){
                DFS( i, g[], vis);
            }
        }
    }
}
```

Quiz 2 :- Time Complexity for DFS ?

Q How many times you are touching each Node ?

↳ 1 time for Each Node

TC → O(N + E) ← No of times in total For loop will run

No of times DFS will run

SC → O(N + (N+E) + N) ← Recursion Stack.

vis

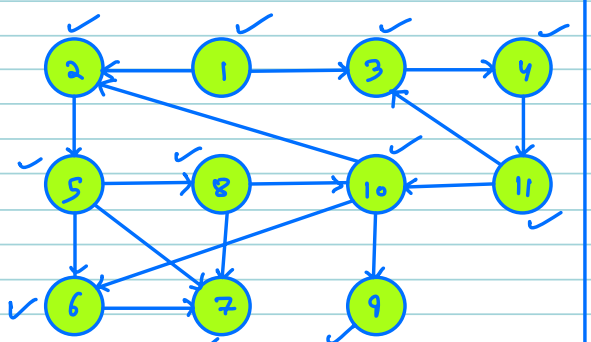
↳ Adjacency List

↳ O(N+E)

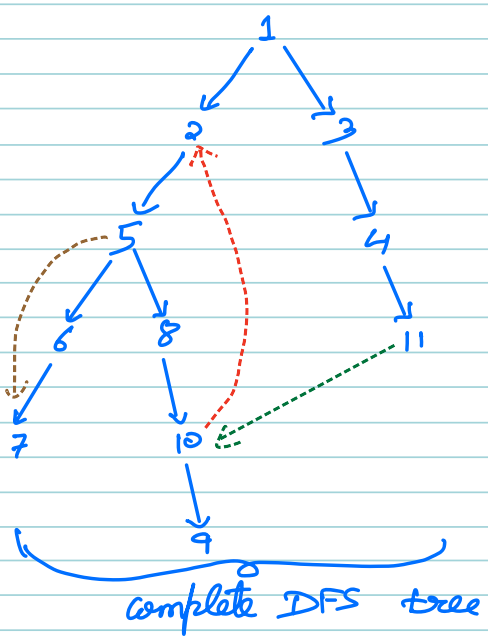
Question :- DFS over Directed graph

vis[12] =

0	1	2	3	4	5	6	7	8	9	10	11
	T	T	T	T	T	T	T	T	T	T	T



DRY RUN



complete DFS tree

{Tree Edges} → DFS Tree

{Forward Edge} → {Any Node → Descendant}

{backward Edge} → {Any Node → Ancestor}

{Cross Edge} → {Nodes not in a Same path}

FACT :- If Backward Edge is present, there is Cycle

Q. Identify If there is a cycle in Directed Graph!

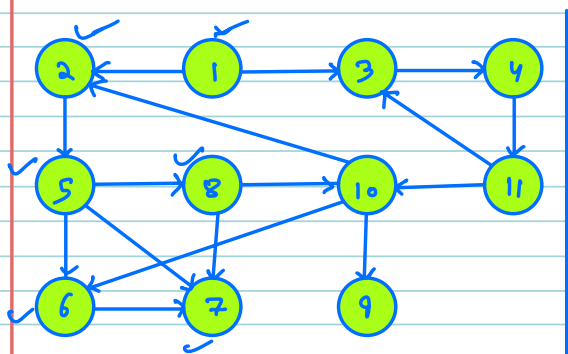
↳ We need to identify Backward Edge.

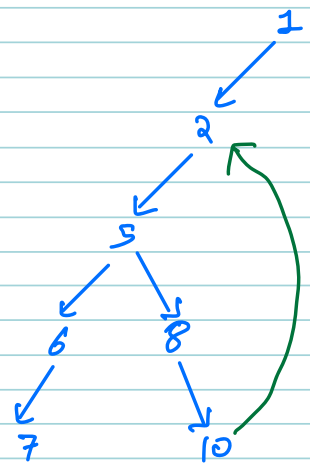
DRY RUN

	0	1	2	3	4	5	6	7	8	9	10	11
Vis[12] =	T	T				T	T	T	T		T	

Stack =	T	T				T			T		T	

↑ Stores Element in path





→ Caller need to insert source in Stack.

PSEUDO CODE

```
bool isCycle( list<int> g[], int s, bool[] vis,
              bool[] st){
    vis[s] = True;
    for( i=0; i<g[s].size(); i++){
        v = g[s].get(i);
        if( st[v] == True){
            return True;
        }
        if( vis[v] == False){
            st[v] = True;
            if( isCycle(g, v, vis, st)){
                return True;
            }
            st[v] = False;
        }
    }
    return False;
}
```

TC — O(N+E)

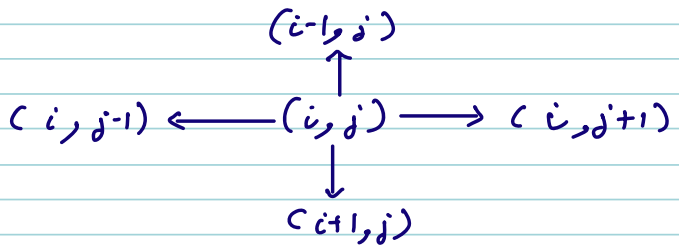
SC — O(N+E)

Question :- No. of Islands ?

Given a matrix of 1's & 0's. Find the No. of Islands present :

mat[][] → 1: land
0: water

Island :- Water in all directions



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

Ex :-

1	1	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	0	0	0
1	0	1	1	1

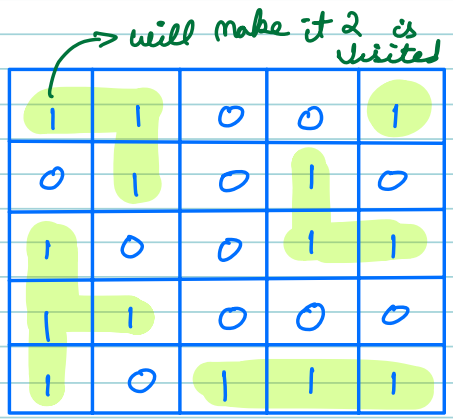
⇒ Total 5 Islands

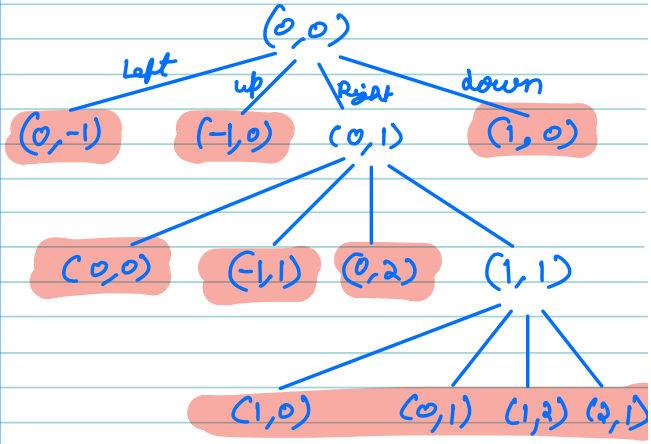
Q How this question is mapping to Graphs ?

Note :- 1.) 1 is indirectly representing nodes.

2.) Adjacency List → Any node can atmax 4 neighbour.

Q How to traverse over such Graph?





⇒ So we will try to form Island from all the cells & see how many Islands are there

PSEUDO CODE

```
int IslandsC int mat[][]){
    int c=0
    for(i=0; i<n; i++){
        for(j=0; j<m; j++){
            if(mat[i][j]==1){
                c++
                DFSC(mat, i, j);
            }
        }
    }
}
```

```
void DFS ( int mat[][], int i, int j){
    if ( i<0 || j<0 || i>=n || j>= m || mat[i][j]==0
                                    || mat[i][j] == 2) {
        return;
    }
    mat[i][j] = 2;
    DFS ( i, j-1);
    DFS ( i-1, j );
    DFS ( i, j+1);
    DFS ( i+1, j);
}
```

TC → O(N * m)

SC → O(N*m) → DFS calls

Question

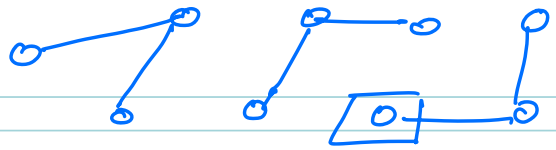
Problem

In the LinkedIn ecosystem, understanding the structure of professional networks is crucial for both users and the platform itself.

A new feature is being developed to visualize the network of a user in terms of "**clusters**". These clusters represent groups of LinkedIn users who are all connected to each other, either directly or through mutual connections, but do not have connections to users outside their cluster. This visualization aims to help users to increase their **Reach**.

Problem Statement

Given **A** denoting the total number of people and matrix B of size Mx2, denoting the bidirectional connections between LinkeIN users, and an Integer C denoting the user ID of [redacted] the target person, find out the number of connections that this person should make in order to connect with all the networks.



=> This is Exactly an Island problem. We just Need to Subtract 1 from no. of cluster present