Graphs - 1

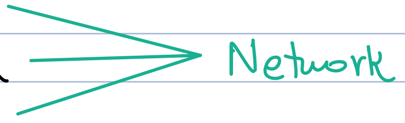
Topics To Cover:

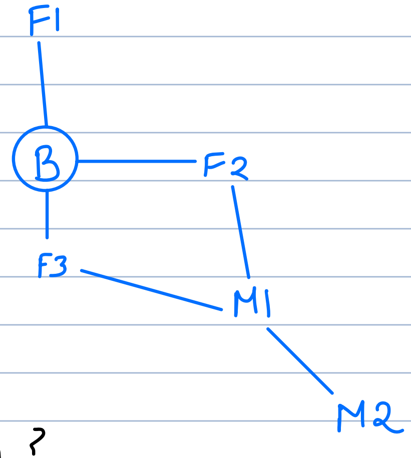
i) Introduction To Graphs
ii) Types of Graphs.
iii) DFS
iv) Detect cycle in a graph
✗ v) No. of islands

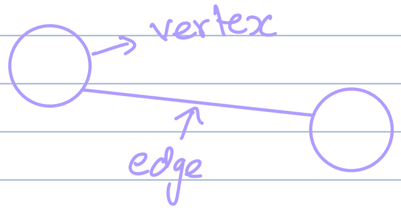
Song: How You Remind Me
- Nickleback.

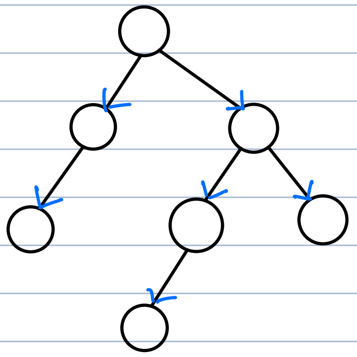
Hi Everyone!!!

Graphs:
- Map
- Facebook
- Airports









1. Is every tree a graph? ↳ Yes

2. Is every graph a tree? ↳ No.

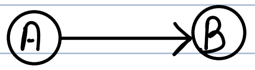
① Tree always has a special node called Root

② If there are N nodes than we have N-1 edges

③ A tree Cannot have a cycle.

Types of Graphs:

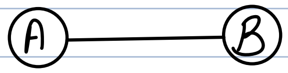
Directed Graphs



A → B ✓

× B → A ×

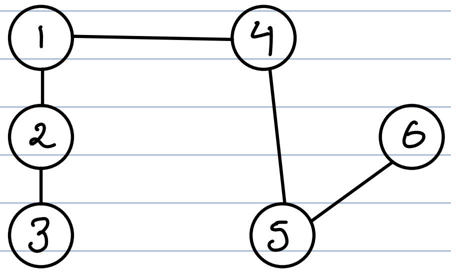
Undirected Graphs



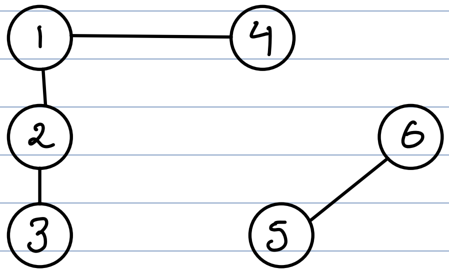
A → B ✓

B → A ✓

Connected Graph

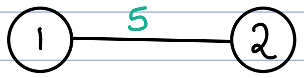


Disconnected Graph

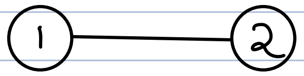


2 components in one graph

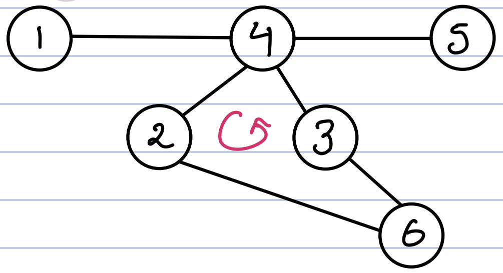
Weighted Graph



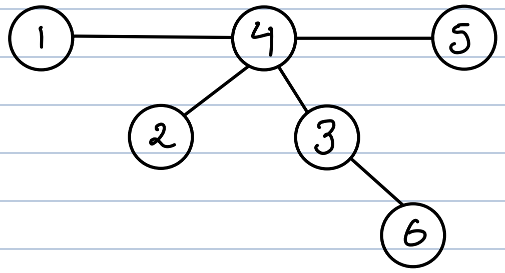
Unweighted Graph

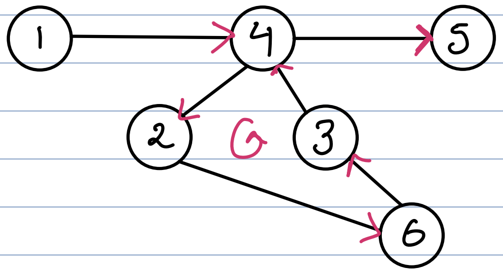


Cyclic Graph

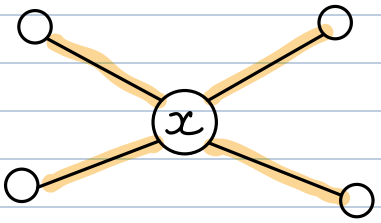


Acyclic Graph

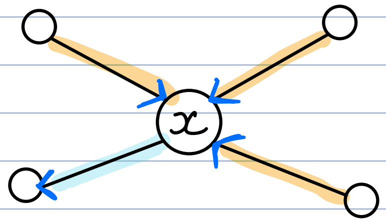




* Degree



degree(x) = 4

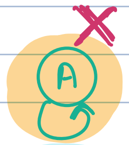


towards Node
↳ indegree : 3

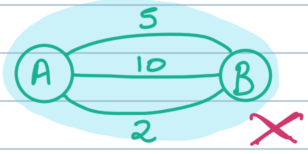
outdegree : 1
away from Node

* Simple Graph

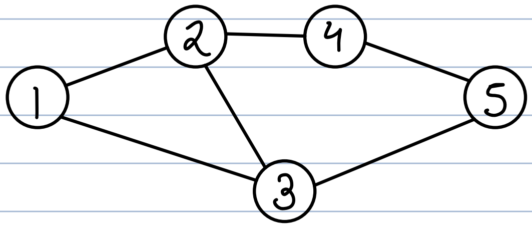
Graph without self loops



or multi edges b/w the same pair of vertices



How to store a graph ?



Adjacency matrix

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

SC : $O(V^2)$

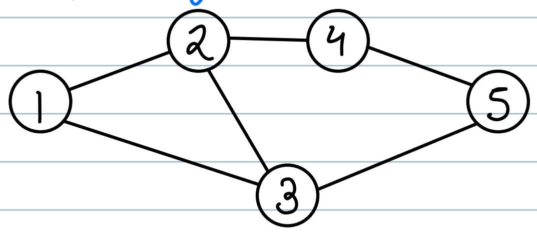
Adv : easy access / update

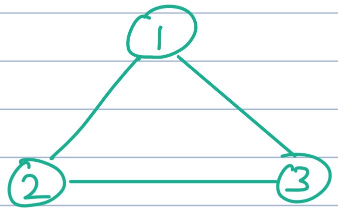
Dis : lot of space

↓ col tells indegree

→ row tells out degree.

② Adjacency list.





0 — { }
1 — {2, 3}
2 — {1, 3, 4}
3 — {1, 2, 5}
4 — {2, 5}
5 — {3, 4}

[V] [2E]

1 — {2, 3}
2 — {1, 3}
3 — {1, 2}

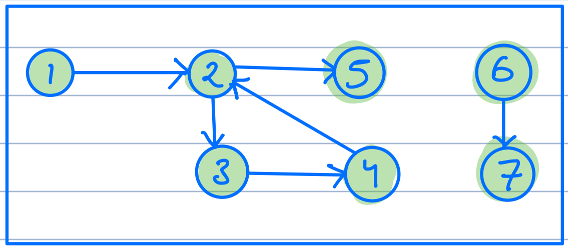
SC : O(V+E)

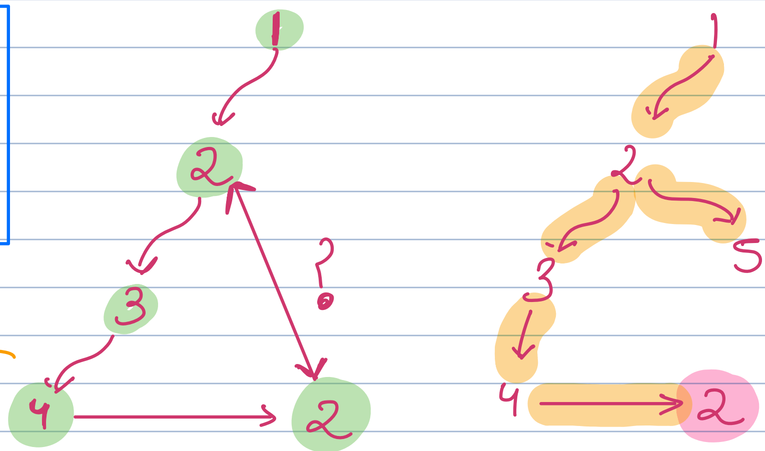
if edges had weight ?

0
1 → {(Node, wt.)}
2

↑ pair class

Depth First Search





solution for cycle

Visited Ar : [F ~~F~~ T ~~F~~ T ~~F~~ T ~~F~~ T ~~F~~ T ~~F~~ T ~~F~~ T]
0 1 2 3 4 5 6 7

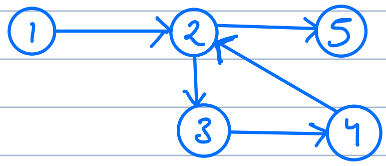
DFS output : ① ② ③ ④ ⑤ ⑥ ⑦

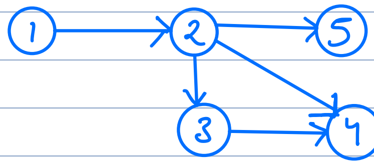
```
fn solve ( Ar <List<list<int>>>)  → Adjacency list
   {
      bool visited [v+1];   # by default each val = False
      for (i = 1 ; i ≤ v ; i++)
        {
           if (visited [i] == False)
             {
                DFS ( Ar, visited , i)
             }
        }
   }

fn DFS (Ar , visited , node)
   {
      print (node)
      visited [node] = True ;
      for (int nbr : Ar [node] )
        {
           if (visited [nbr] == False)
             {
                DFS (Ar, visited , nbr);
             }
        }
   }
```

Break 10:39 → 10:42

Song : Skyfall
— Adele

Q. Check if given directed graph has a cycle or not.



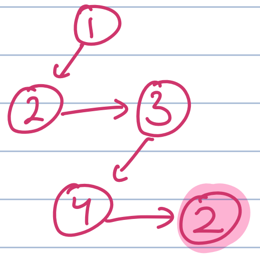


visited :	F	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	F
	0	1	2	3	4	5

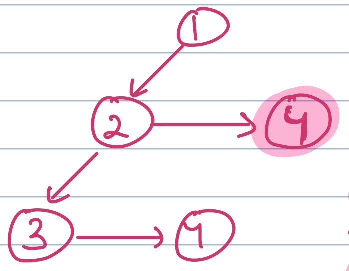
↑ (under 2)

visited :	F	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	F
	0	1	2	3	4	5

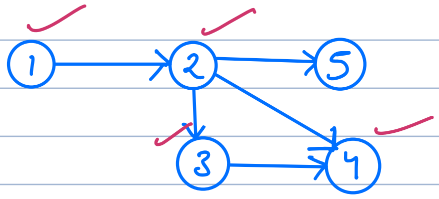
↑ (under 4)



∴ cycle !!!



∴ ~~cycle~~



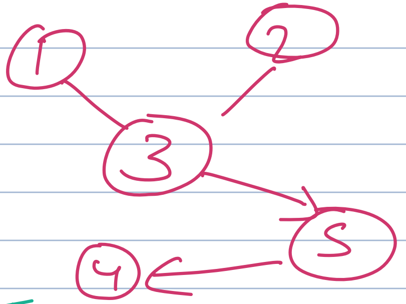
① → ② → ③ → ④

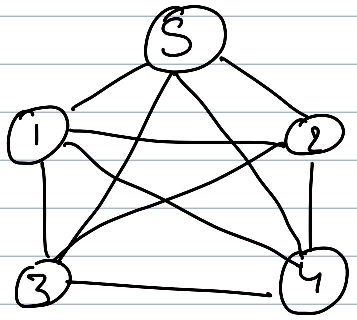
① → ② → ④

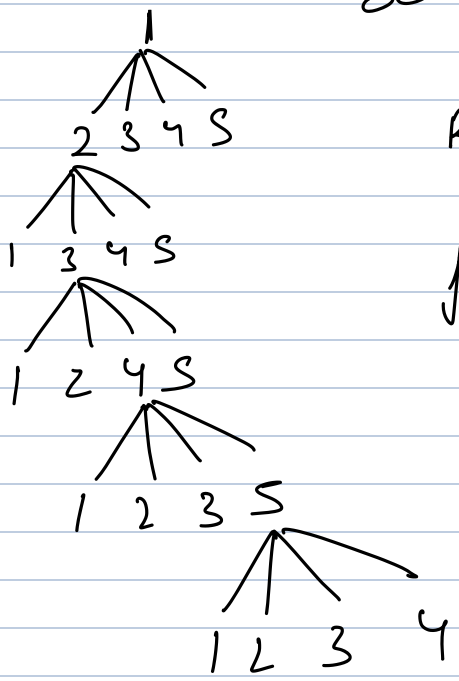
path : [	F	T	T	T	T	F]
	0	1	2	3	4	5

```
fn solve ( Ar < List < list<int>>>> )    → Adjacency list
{
    bool visited [v+1];   # by default each val = False

    bool path [v+1];
    for (i = 1 ; i ≤ v ; i++)
    {
        if (visited [i] == False)
        {
            DFS (Ar, path, visited, i);
        }
    }
}
```

```
fn DFS (Ar, path, visited, node)
{
    visited [node] = True ;
    path [node] = True ;
    for (int nbr : Ar [node])
    {
        if (path [nbr] == True) return True ;

        if (visited [nbr] == False)
        {
            if ( DFS (Ar, path, visited, nbr) == True)
                return True ;
        }
    }
    path [node] = False ;

    return False
}
```







TC: $\Theta(V+E)$

SC: $\Theta(V)$

↳ path + visited + stack size

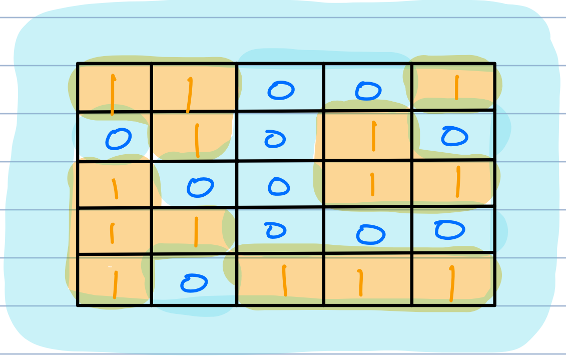
Recursive calls = 5 = V

for loops = 20

↓

2E → 10

Number of Islands



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

0 → water
1 → land

Ans = 5

idea 1: Number of Components.

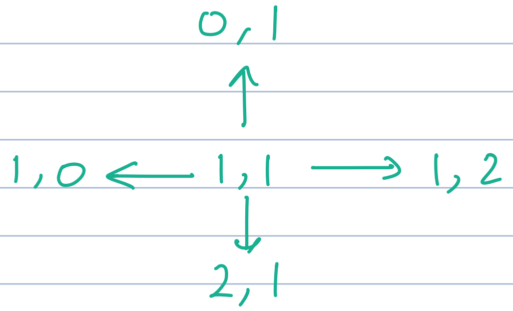
$$Map = \begin{bmatrix} -1 & -1 & 0 & 0 & -1 \\ 0 & -1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 & -1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

```
islands = 0

for (i=0 ; i<N ; i++)
    for (j=0 ; j<M ; j++)
    {
        if (Map[i][j] == 1)
        {
            islands++;

            DFS(Map, i, j)
        }
    }
}
```

dx = [0 1 0 -1]
dy = [1 0 -1 0]

i, j+1   i+1, j   i, j-1   i-1, j

```
fn DFS(Map, i, j)
{
    Map[i][j] = -1;

    dx = [0 1 0 -1]
    dy = [1 0 -1 0]

    for (k=0 ; k<4 ; k++)
    {
        Ni = i + dx[k];
        Nj = j + dy[k];

        if (Ni ≥ 0 && Nj ≥ 0 && Ni < N && Nj < M
                && Map[Ni][Nj] == 1)
        {
```



```
            DFS (Map, Ni, Nj);
        }
    }
}
```

TC: $O(N*M)$

SC: $O(1)$ / $O(N*M)$

↑ in case you cannot change input.