Congratulations for maintaining >80% PSP 🔥🥳

Deepak Dharani	Gobika K
Divya Shanmugam	Vinod Supnekar
Shubham Verma	Nandini Umbare
Zahra Sidhpuri	Mariappan Subramanian
SANTOSH KUMAR SHARMA	Saurabh Vishwakarma
Rakesh	Lakshmi Sravani
Pavan Satish	Devender swami
Rajesh Rohan	Rudrakshi Srivastava
SANDEEP MUDAPAKA	christon cardoza
Santosh Nase	Damini
Ajay Jain	Anindya
Arijit Dutta	Subhashish B
Vitul Gupta	Rajkumar
Mrudang Vora	Gujjula Samara simha reddy
Brian Sam Varghese	Manas
Manikanta sai	Vishal Patel

Today's Content →

Introduction to Graphs

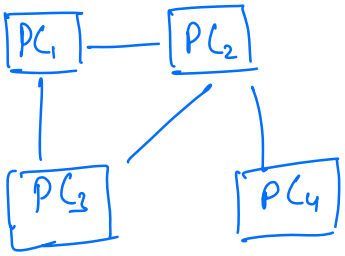
Types of Graphs

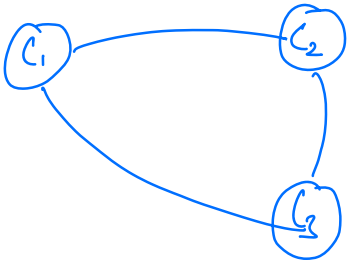
DFS

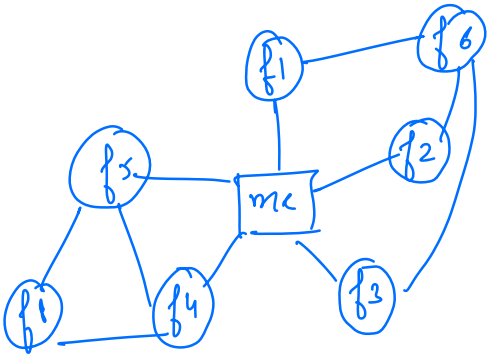
BFS

Detect cycle in directed graph.

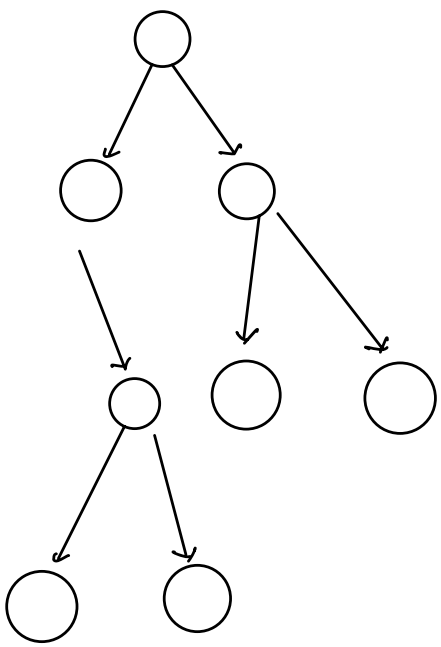
Graph → network.







graph → collection of nodes (or vertices) and edges.



① Tree always has root node.

② N nodes → N-1 edges.

③ Cycle can't be there in tree.

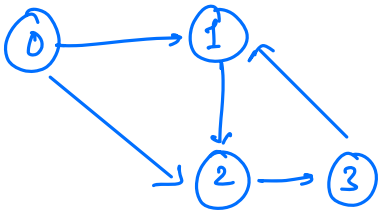
Every tree is a graph. ✓

Every graph is a tree. ✗

⇓

[Tree is a subset of Graph]

How to store graphs?



4 Nodes (N)
5 edges (E)

① Adjacency Matrix

	0	1	2	3
0	0	1	1	0
1	0	0	1	0
2	0	0	0	1
3	0	1	0	0

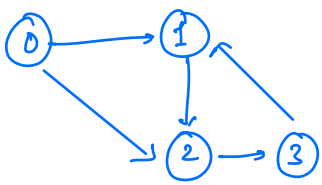
mat[i][j] = 0 ⇒ There is no edge from i to j.

mat[i][j] = 1 ⇒ There is an edge from i to j.

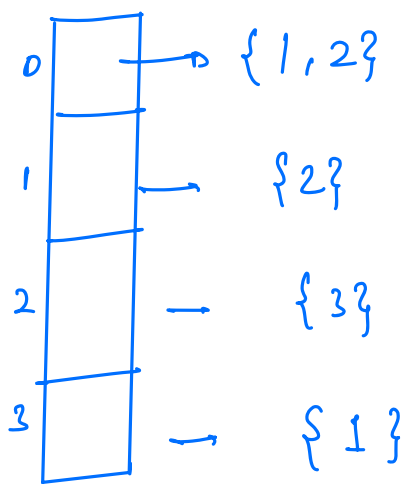
SC → $O(N^2)$

② Adjacency List

(Array of lists)



N = 4
E = 5



A[i] → adjacent nodes in the list
↳ neighbours of i in the list.

$\begin{bmatrix} 1 \le N \le 10^5 \\ 1 \le E \le 10^5 \end{bmatrix}$

SC → $O(N+E)$

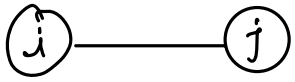
Properties / Types of Graph

① **Directed**

i to j ✓

j to i ✗

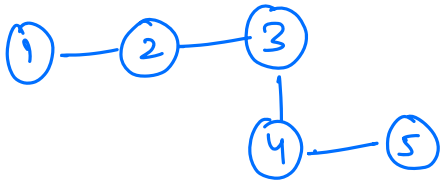
Un-directed Graph (Bi-directional)



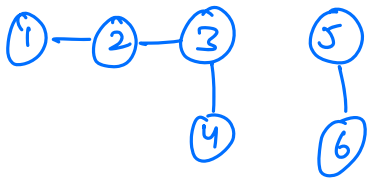
i to j ✓

j to i ✓

② **Connected**



Disconnected



③ **Weighted**

Un-weighted

mat[i][j] = 0 if there is no edge from i to j

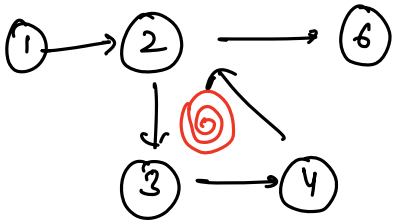
= W_{ij} if there is edge from i to j.

graph[i] = { {1, 5}, {2, 3}, {4, 7} - - }

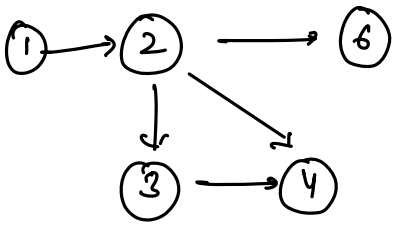
list of pairs

pair {
  vtx :
  edge wt ;
}

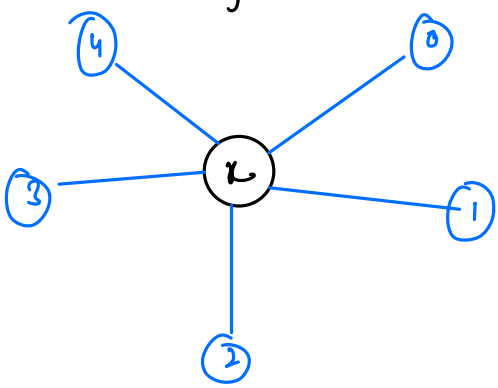
(4) Cyclic.



Acyclic



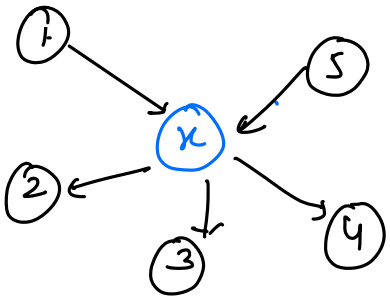
(5) Degree.



degree(x) = 5

No. of edges connected with a node.

In-degree / out-degree



in-degree(x) → incoming edges [2]

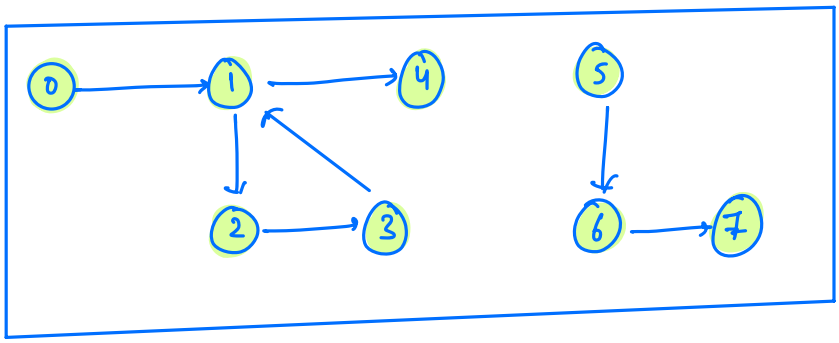
out-degree(x) → outgoing edges [3]

N nodes. → numbered from 0 to N-1

if they are not numbered from 0 to N-1, then we will do the mapping and get the order from 0 to N-1.

Traversals

① Depth First Traversal (Pre-order traversal of tree)

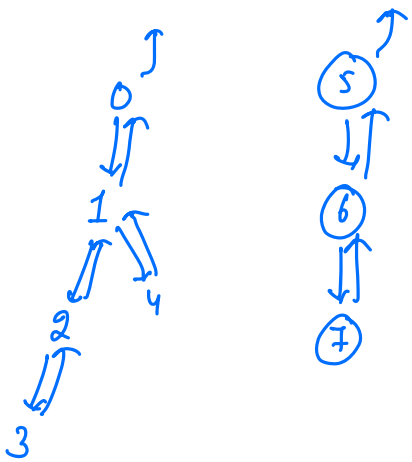


N = 8 , E = 7

→ Keep track of visited nodes.

visited →

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



code →

```
//Graph → given

boolean visited[N];   // ∀i, visited[i] = false

for( i=0; i<N; i++){
    if( visited[i] == false){
        dfs( graph, i, visited);
    }
}
```

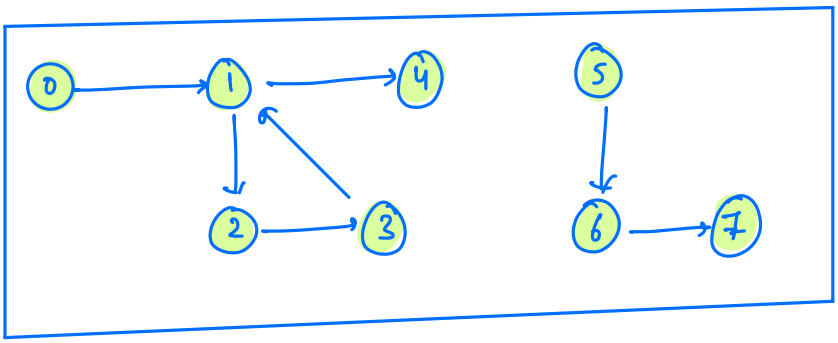
```
void dfs( Graph, src, visited[N]){
    print(src);
    visited[src] = true;
    for( int nbr : graph[src]){
        if( visited[nbr] == false){
            dfs( graph, nbr, visited);
        }
    }
}
```

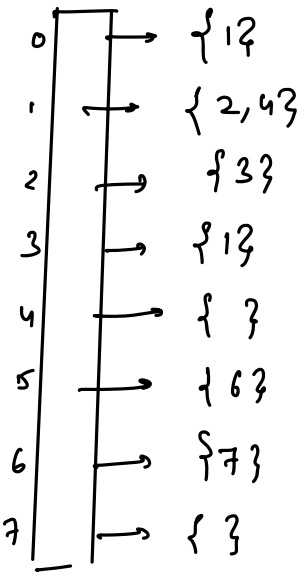
T.C → O(N+E)
S.C → O(N)

↓

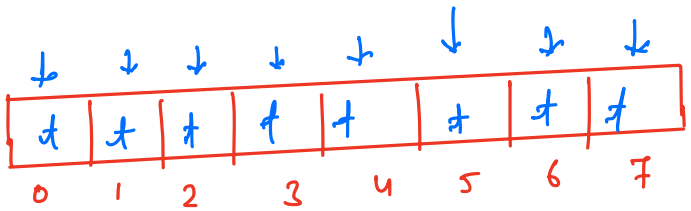
visited[N] + max size of stack.

Breadth First Traversal → level order traversal.

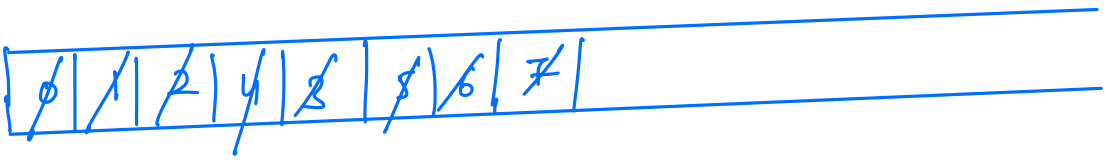




visited →



queue →



print → 0, 1, 2, 4, 3, 5, 6, 7

code →

```
//Graph → given

boolean visited[N];   // ∀i, visited[i] = false

for( i=0; i<N; i++){
    if( visited[i] == false){
        bfs( graph, i, visited);
    }
}
```

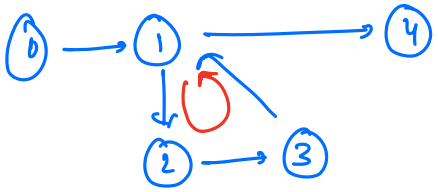
```
void bfs( graph, src, visited){
    Queue<int> q;
    q.enqueue(src), visited[src]= true;
    print(src);
    while( q.isEmpty() == false){
        vr = q.dequeue();
        for( int nbr : graph[vr]){
            if( visited[nbr] == false){
                visited[nbr] = true; print(nbr);
                q.enqueue( nbr);
            }
        }
    }
}
```

T.C → O(N+E)
S.C → O(N)

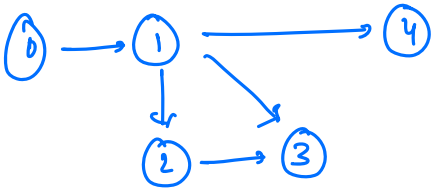
↓

visited [N],
Queue

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



ans → true.



ans → false

→ if a visited node is encountered again → cycle. ✗

→ if a visited node in current path → cycle ✓
is encountered again

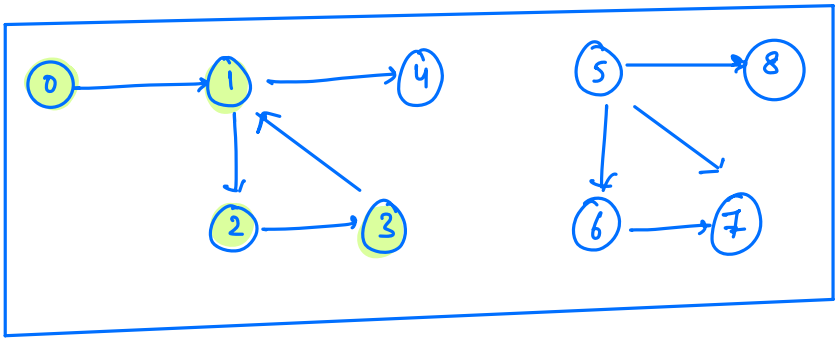
code →

```
boolean visited [N], // ∀i, visited [i] = false;
boolean path [N] , // ∀i, path [i] = false;

for( i = 0; i < N; i++){
    if( visited [i] == false){
        if ( dfs ( graph, i, visited, path) == true){
            return true;
        }
    }
}
return false;
```

```
boolean dfs ( Graph , src , visited[N], path[N] ){
    visited[src] = true;
    path[src] = true;
    for( int nbr : graph[src]){
        if( path[nbr] == true){ //if nbr is already present
                                //in current path
            return true;
        }
        if( visited[nbr] == false && dfs(graph, nbr, visited, path)){
            return true;
        }
    }
    path[src] = false; //removing src from curr path before
                       //returning from src.
    return false;
}
```

T.C → O(N+E)
S.C → O(N)

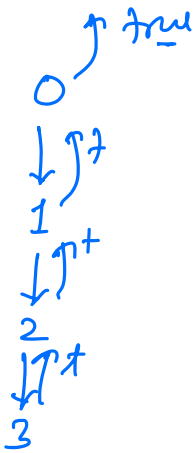


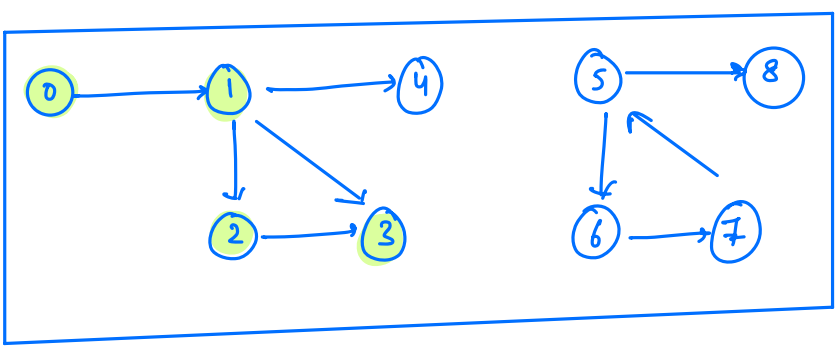
visited [ ] →

0	1	2	3	4	5	6	7	8	9
t	t	t	t	f	f	f	f	f	f

path [ ] →

t	t	t	t	f	f	f	f	f	f





60-70% → BFS, DFS