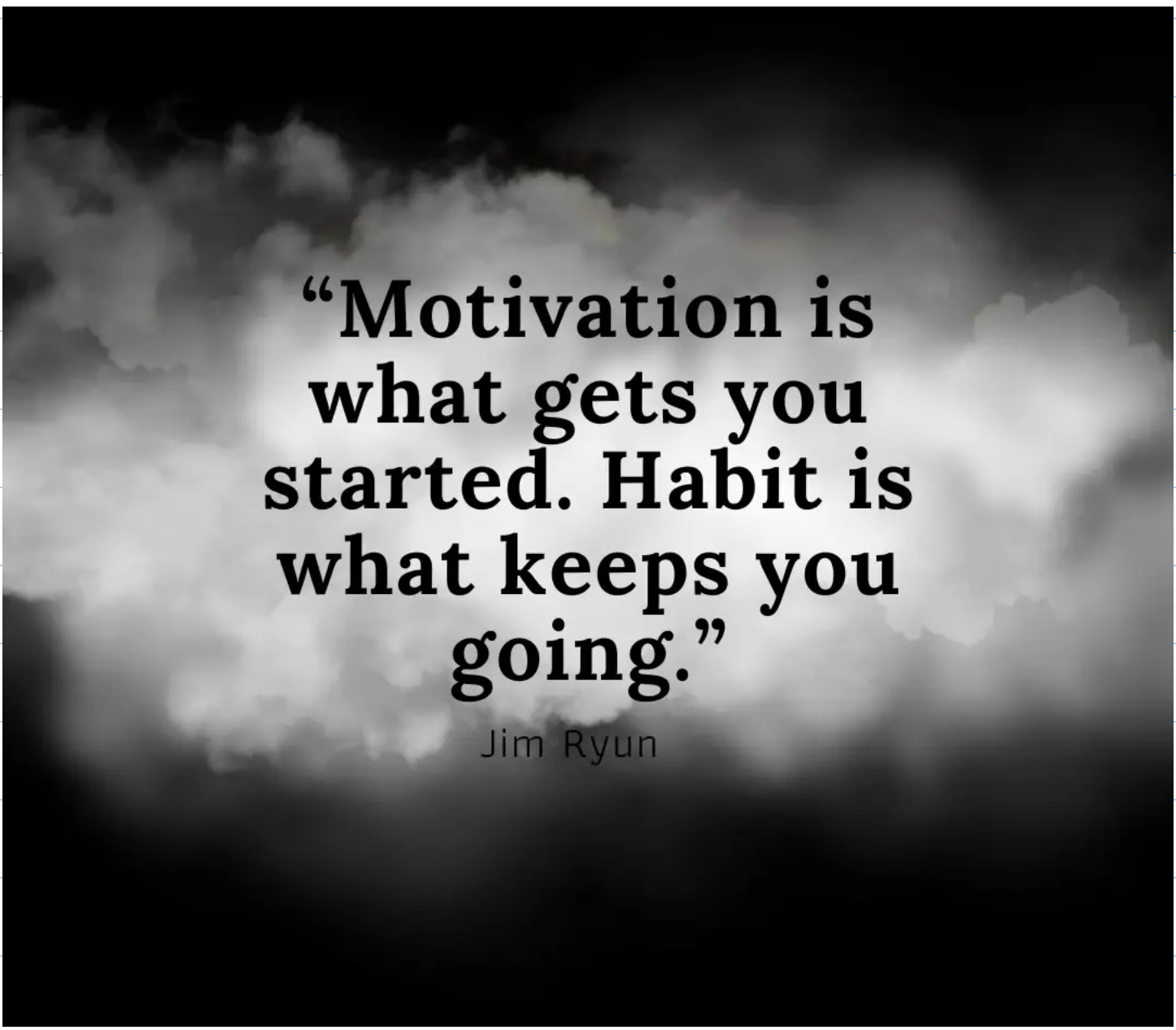
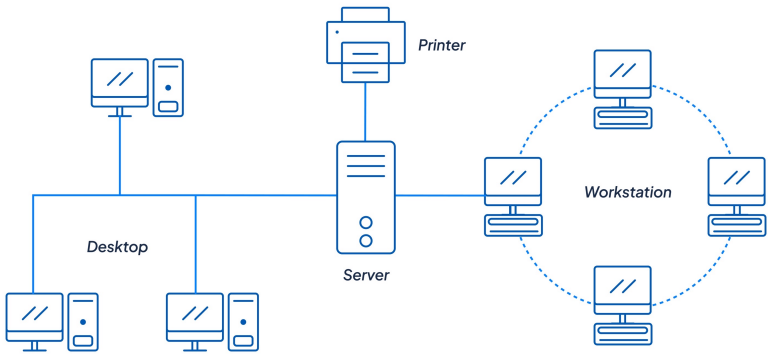
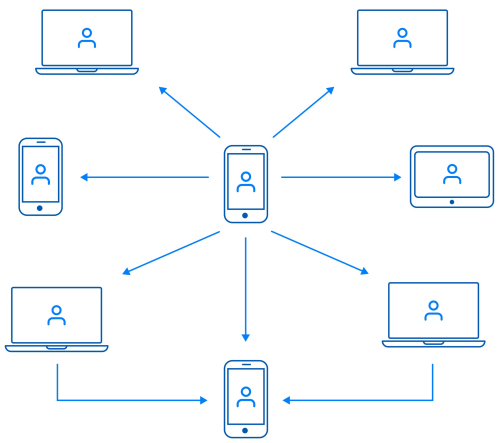
Graphs 1 : Introduction, DFS & Cycle Detection

TABLE OF CONTENTS



Graphs

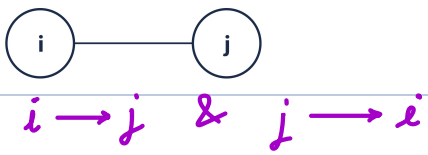




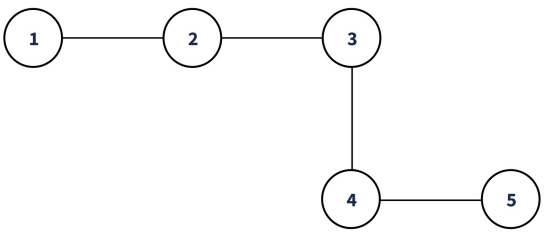
Properties / Types of Graphs

1. Directed

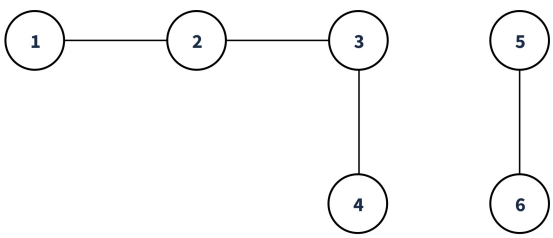
Un-directed graph



2. Connected

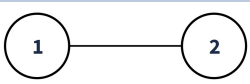


Disconnected

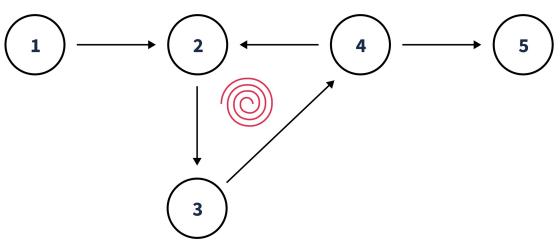


3. Weighted

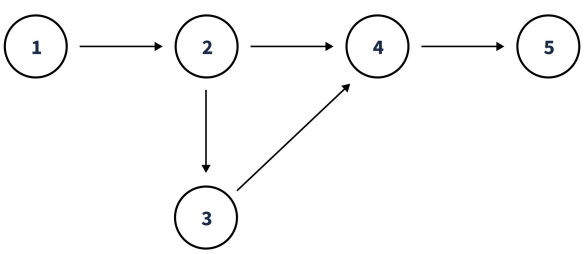
Unweighted

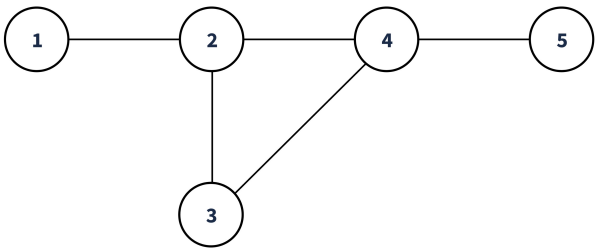


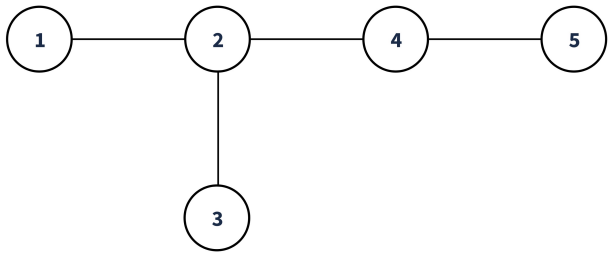
4. Cyclic



Acyclic

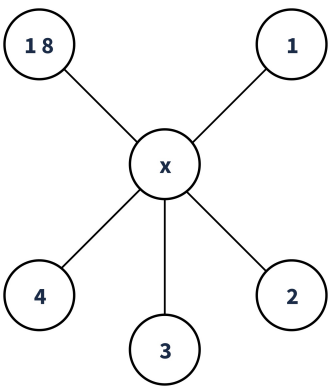






For undirected graph min 3 node cycle ⇒ cyclic graph.

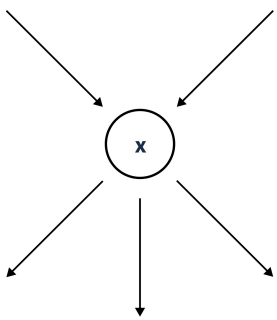
5. Degree



degree(x) = 5

edges connected

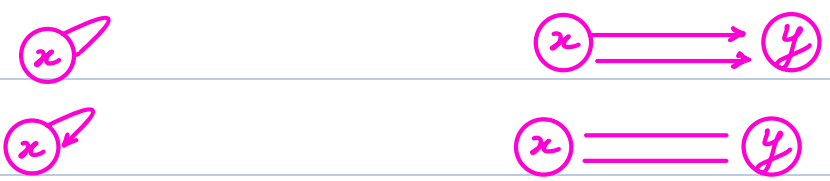
In-degree and Out-degree

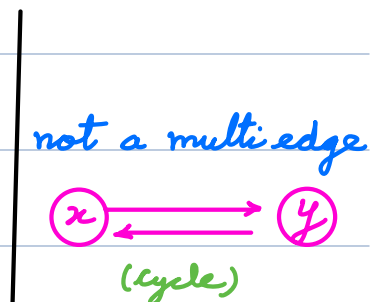


in-degree(x) = 2 incoming edges

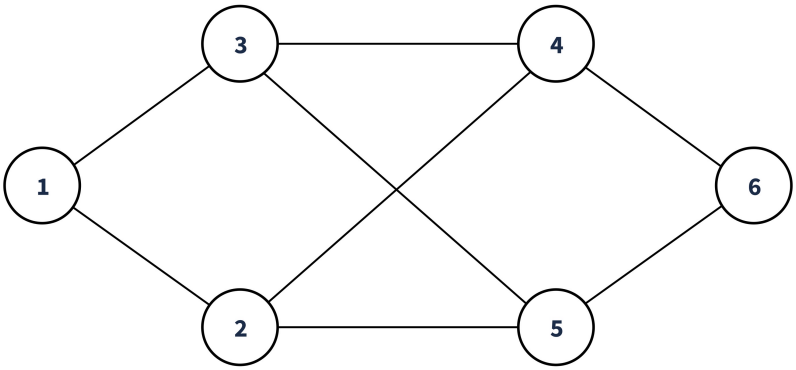
out-degree(x) = 3 outgoing edges

6. Simple Graph → A connected graph without self loops & multiedges.





How to store a graph?



nodes → N = 6

edges → E = 8

1. Adjacent Matrix

N * N matrix

Unweighted

A[i][j] → 1 → i → j

→ 0 → no edge

	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

Weighted

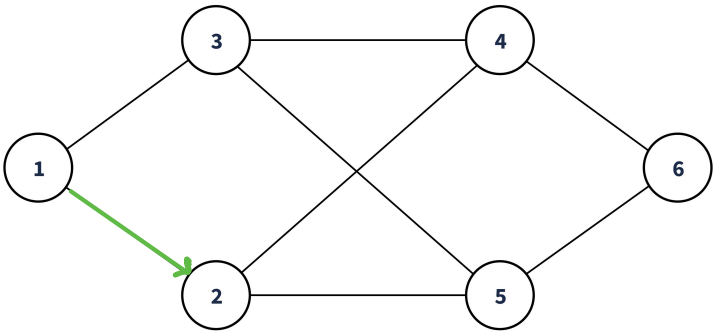
A[i][j] → wt → i —wt→ j

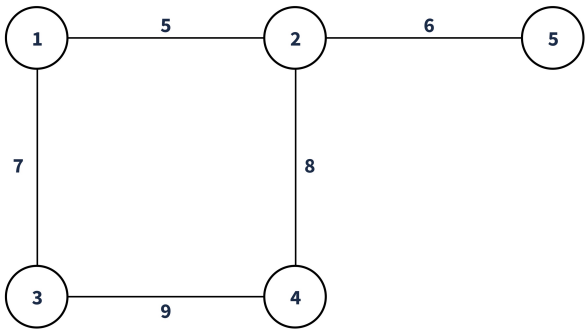
→ 0 → no edge

SC = O(N²)

2. Adjacent List

A[i] → list of nodes connected to i.





outgoing edges in case of directed graph.

0	
1	{2, 3}
2	{~~1~~, 4, 5}
3	{1, 4, 5}
4	{2, 3, 6}
5	...
6	...

0	
1	{(2,5), (3, 7)}
2	{(1,5), (4,8), (5,6)}
3	{(4,9), (1, 7)}
4	...
5	...

A[i] → list of pairs (j, w) i.e. $i \xrightarrow{w} j$

SC = O(N + E) — sum of length of all lists

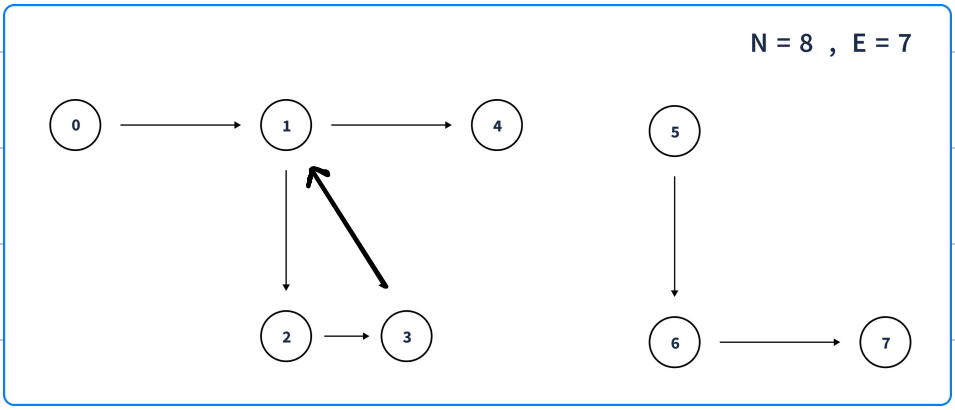
N <= 10^5 } use Adj. List

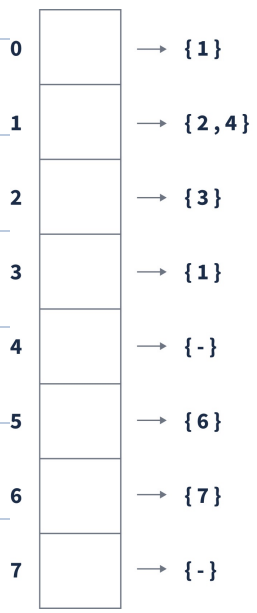
E <= 10^5

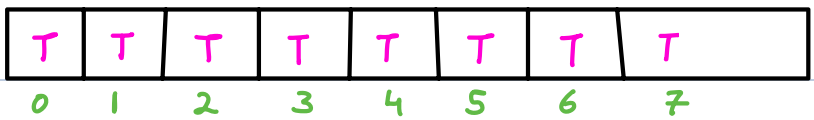
Traversal

Preorder / Inorder / Postorder

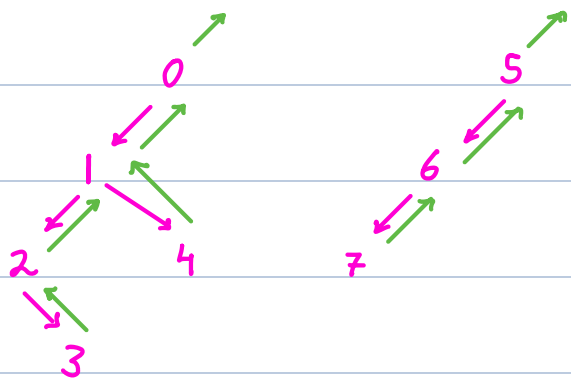
Depth First Traversal







Keep track of visited nodes.



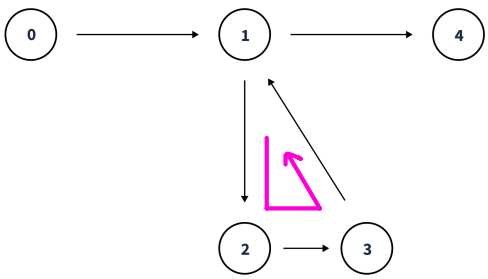
o/p → 0 1 2 3 4 5 6 7

```
∀i , vst[i] = false
for i → 0 to N {   // 1 → N
    if (!vst[i])   dfs(i)
}
```

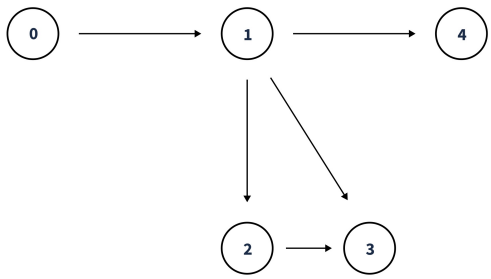
```
void dfs(i) {
    vst[i] = true
    print(i)
    for ( j : Adj[i] ) {   // i → j
        if (!vst[j])   dfs(j)
    }
}
```

TC = O(N+E) SC = O(N) recursion + vst[]

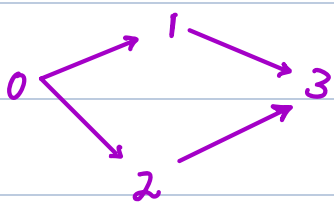
< *Question* > : Check if given directed graph has a cycle or not.



Ans = true



Ans = false



Ans = false

Cycle is present if we visit the same node again in the current path.

path[i] → True if node is in current path.

∀i, vst[i] = false ∀i, path[i] = false

```
for i → 0 to N {  // 1 → N
    if (!vst[i]) dfs(i)
}
```

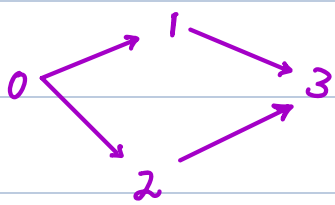
```
boolean dfs(i) {
    vst[i] = true
    path[i] = true

    for ( j : Adj[i] ) {   // i → j
        if (path[j]) return true
        if (!vst[j] && dfs(j)) return true
    }

    path[i] = false
    return false
}
```

TC = O(N+E)

SC = O(N + N + N) = O(N)



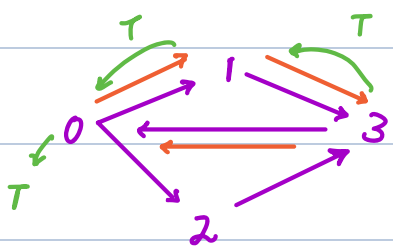
vst →

0	1	2	3
T	T	T	T

path →

0	1	2	3
F	F	F	F

Ans = false



vst →

0	1	2	3
T	T	F	T

path →

0	1	2	3
T	T	F	T

Ans = true