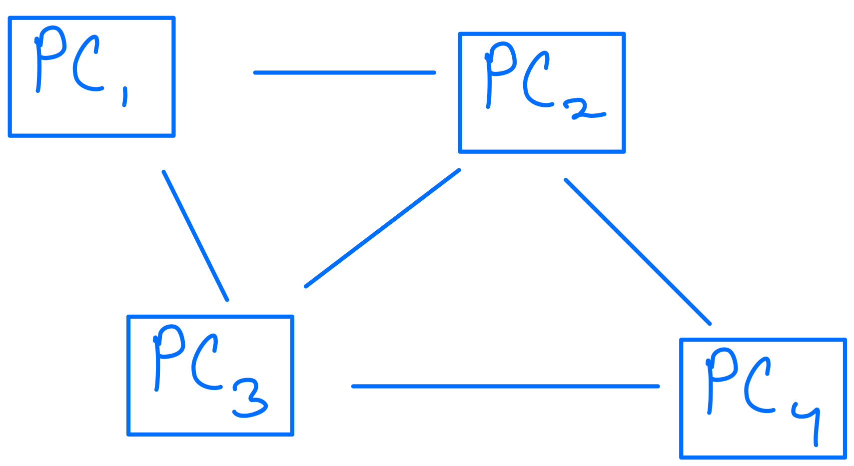
Today's Agenda:-

1. Introduction To Graphs
2. Types of Graphs
3. DFS
4. BFS
5. Detect cycle in a directed graph.

Graphs → Collection of nodes (vertices) & edges.

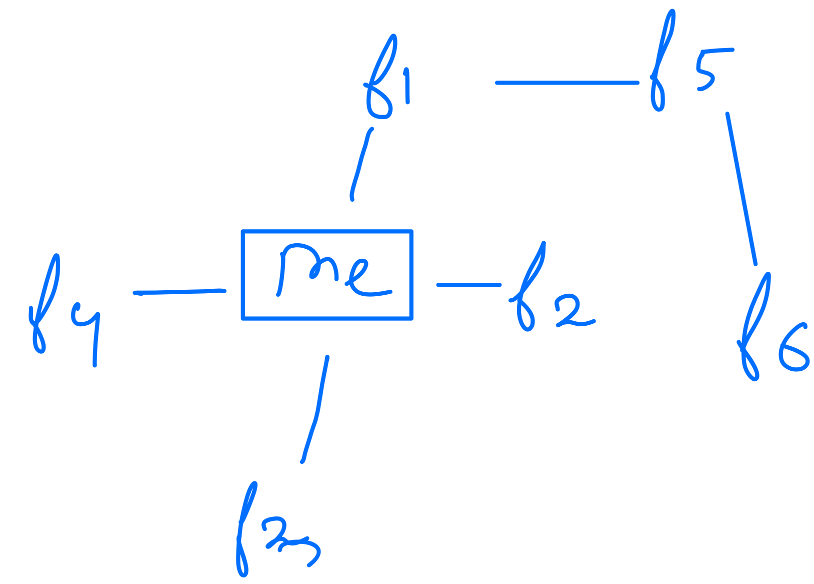
Ex:-1

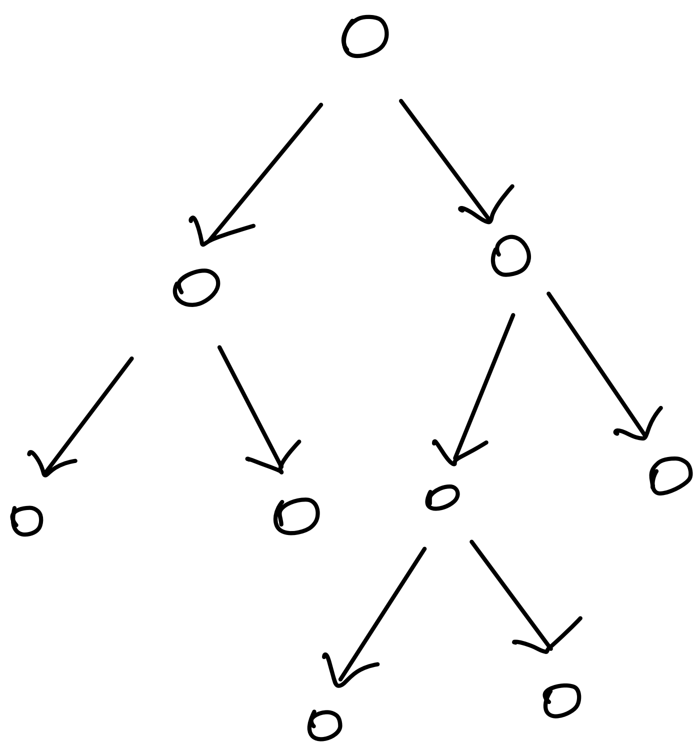
A network of PCs



Ex:-2

Social media





(1) Every tree is a graph

(2) Every graph is a tree.

(Tree is a subset of graph).

1. Tree always has one root.
2. Tree cannot have a cycle.
3. Every node in a tree has a single parent.

How to store graphs ?

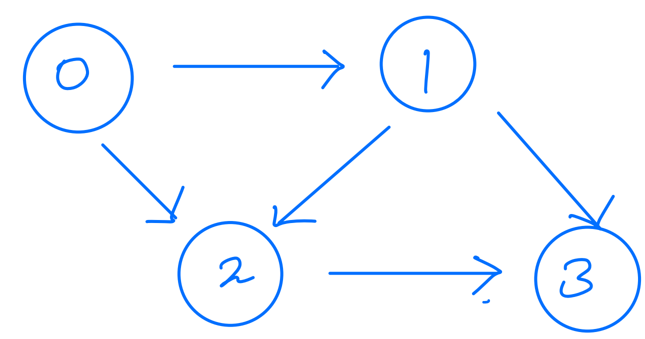
$N = 100$ Nodes.
$E = 150$.

(1) Adjacency matrix.

4 Nodes.
5 Edges.

$N > 10^3$

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

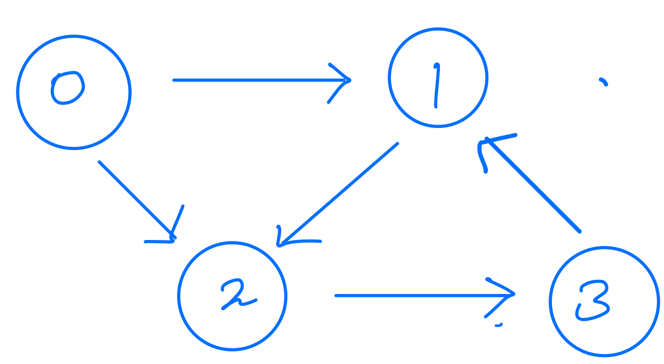


$$mat[i][j] = \begin{cases} 1 ; & \text{if there is an edge from } i \text{ to } j. \\ 0 ; & \text{otherwise.} \end{cases}$$

$O(N^2)$.

(2) Adjacency List.

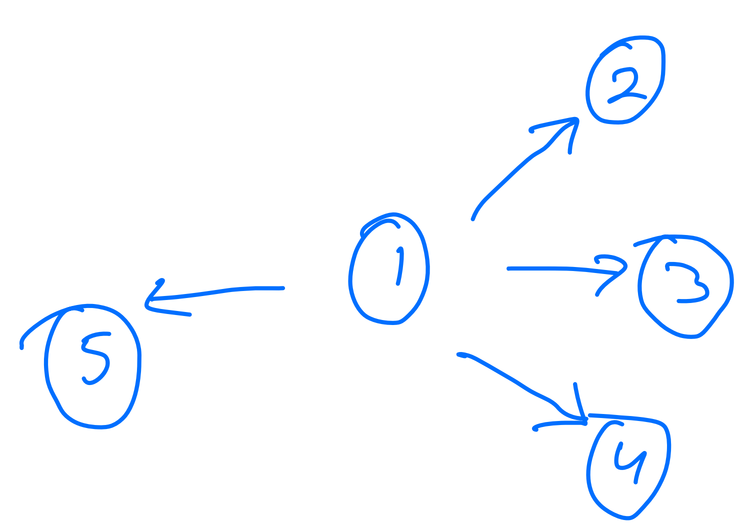
Array of List.



0 → {1, 2}.
1 → {2};
2 → {3}.
3 → {1}.

vector <vector <int>>;

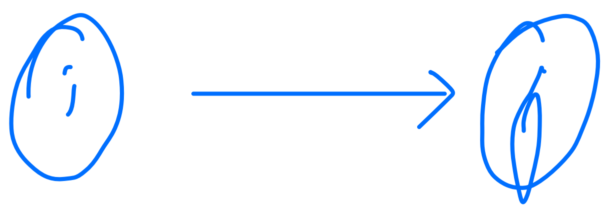
1 → {2, 3, 4, 5}.
2 → {}
3 → {}
4 → {}
5 → {}.



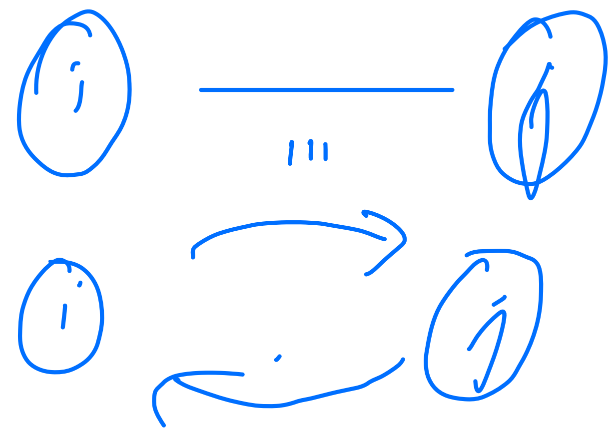
$O(N + E)$.

Properties / Types of Graphs

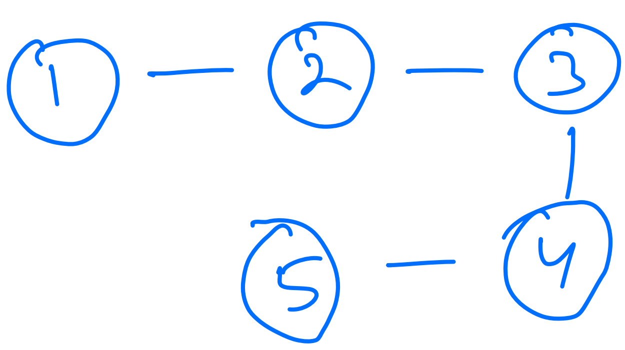
1. Directed



Un-directed graph. (Bi-directional)

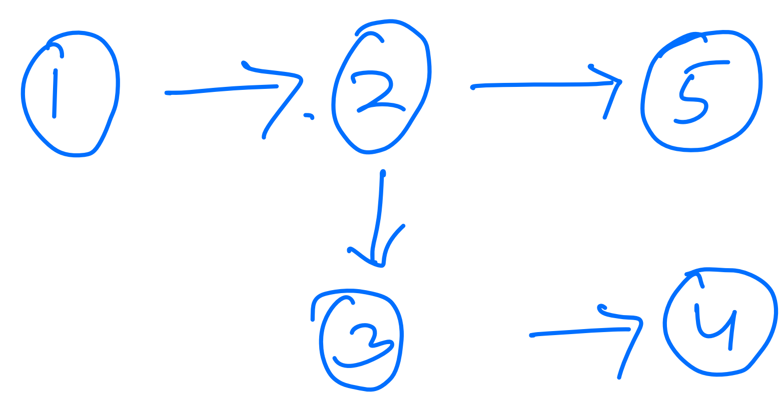


2. Connected



Any two nodes are reachable from each other.

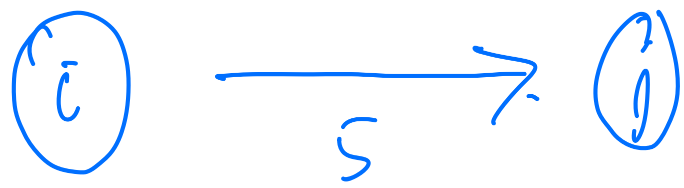
Disconnected



4 — 5 ✗

3 — 2 ✗

3 Weighted

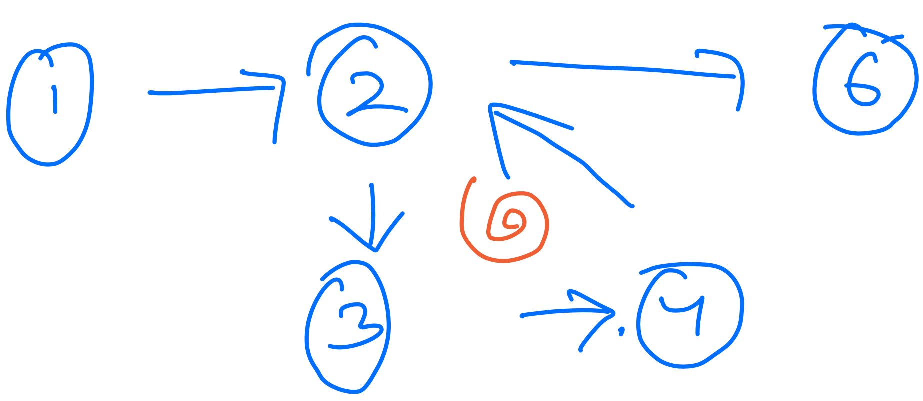


Unweighted

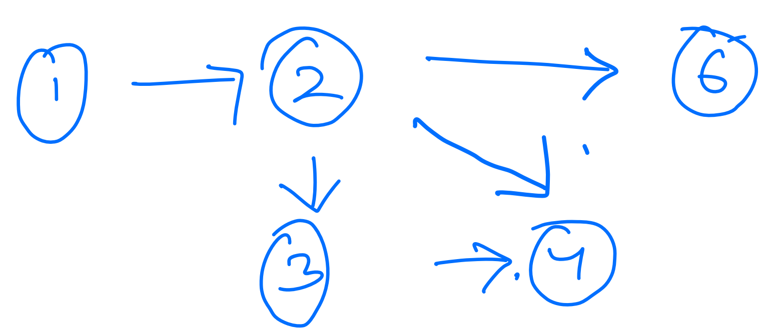
$$\text{mat}[i][j] = \begin{cases} 0, & \text{no edge from } i \text{ to } j \\ W_{ij}, & \text{weight of edge from } i \text{ to } j. \end{cases}$$

$$\text{graph}[i] \rightarrow \underbrace{\{\ \{1,5\},\ \{2,7\},\ \{3,2\}\ \}}_{\text{List} < \text{Pair} >}$$

4 Cyclic

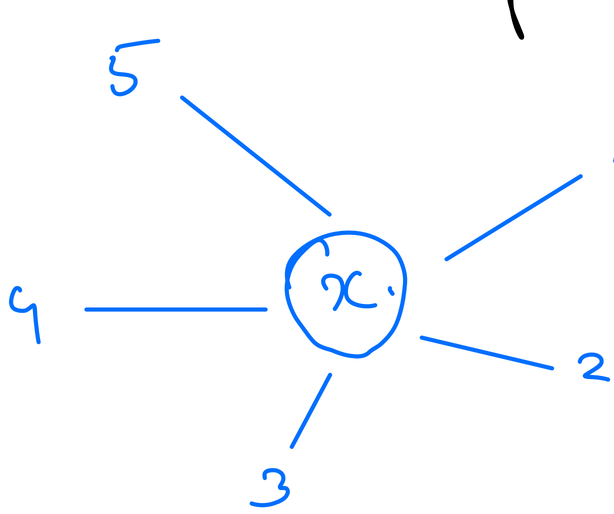


Acyclic.



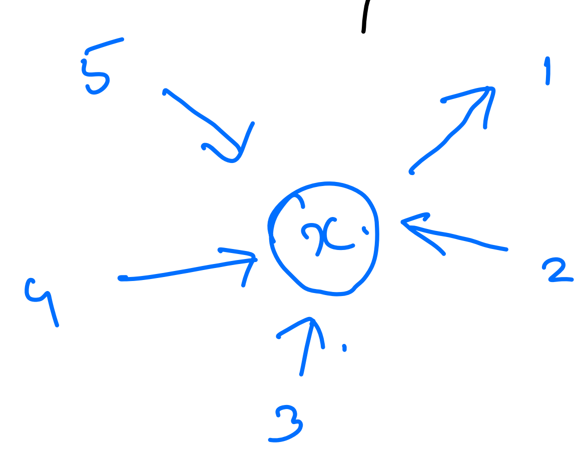
5 Degree

of a node.



$\deg(x) = 5.$

Indegree / Outdegree



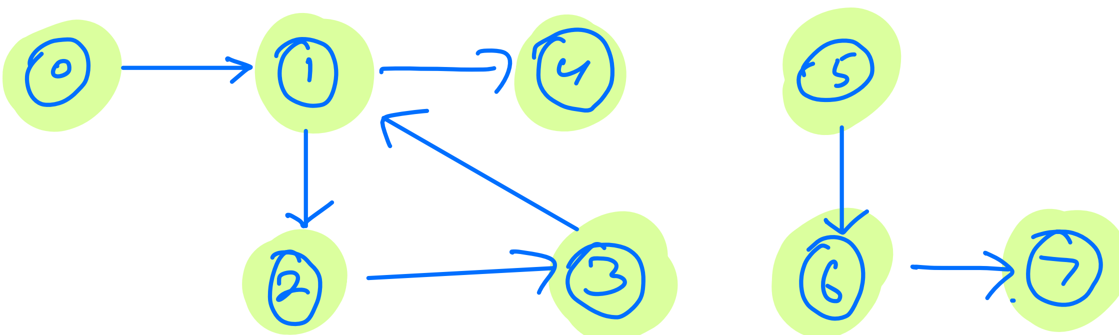
Indegree = 4.

Outdegree = 1.

Traversals

(Level Order Traversal)

1. Breadth First Search (BFS)



0		- {1}
1		- {2, 4}
2		- {3}
3		- {1}
4		- { }
5		- {6}
6		- {7}
7		- { }

visited

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

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

Print:

0 1 2 4 3 5 6 7

vector <vector <int>> graph;

Code:

Graph - given.

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

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

T.C → O(N + E)

S.C → O(N)

```
void bfs(graph, src, visited) {
    Queue<int> q;
    q.enque(src);   print(src);
        visited[src] = true;

    while(q.isEmpty() != true) {
        x = q.deque;
        for(int nbr : graph[x]) {
            if(visited[nbr] == false) {
                q.enque(nbr);
                visited[nbr] = true;
                print(nbr);
            }
        }
    }
}
```

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

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

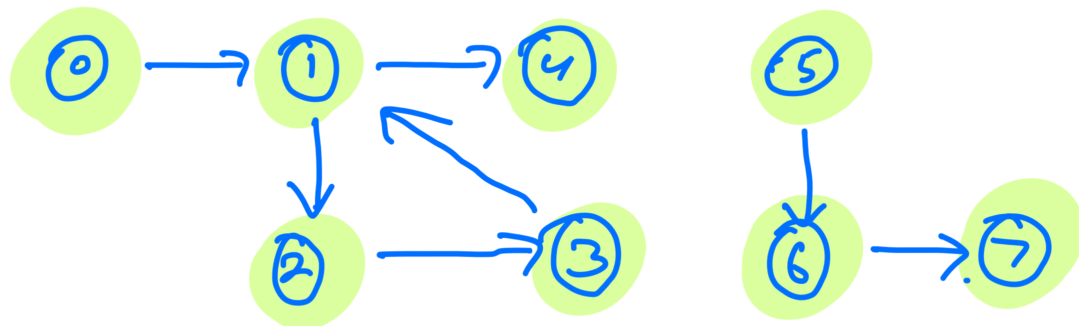
$x = \cancel{0}\ \cancel{1}\ \cancel{2}\ \cancel{4}.\ 3.$

Print

0 1 2 4 . 3 .

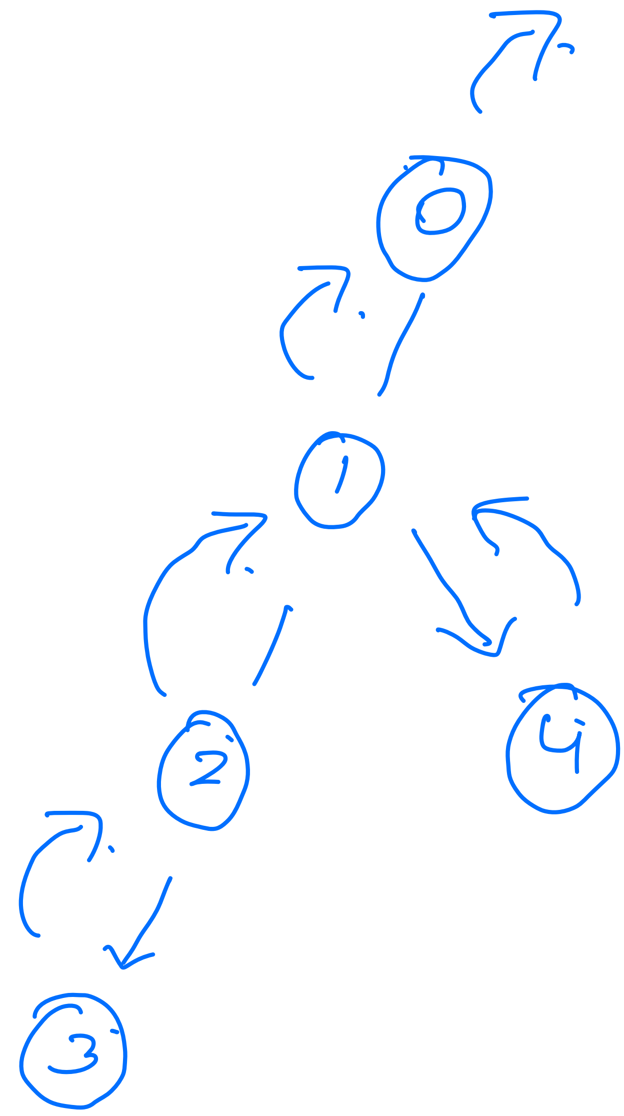
8:50.

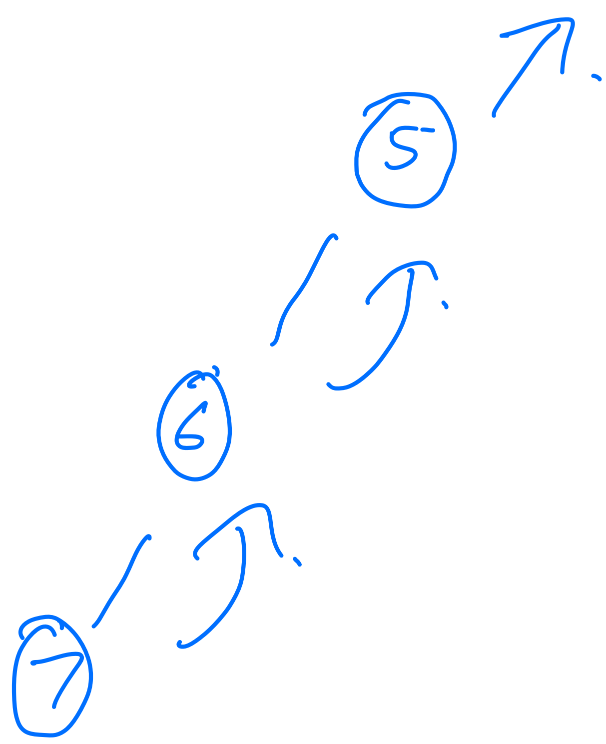
2. Depth First Search (DFS)



visited:

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






```
boolean visited[N] ; ∀i visited[i] = false;

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

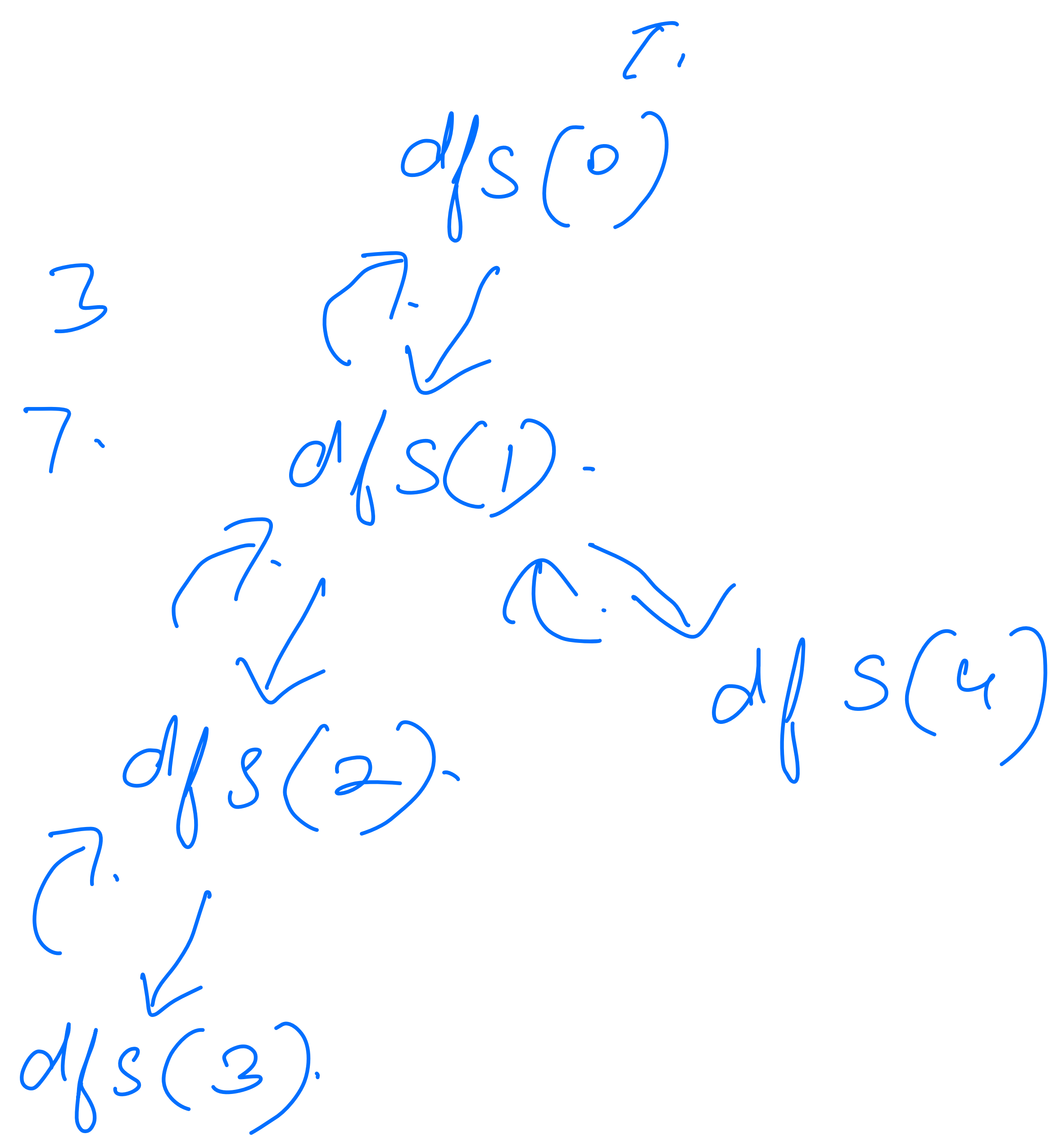
```
void dfs( graph, src, visited ) {

    print (src);
    visited [src] = true;

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

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

Print

0 1 2 3

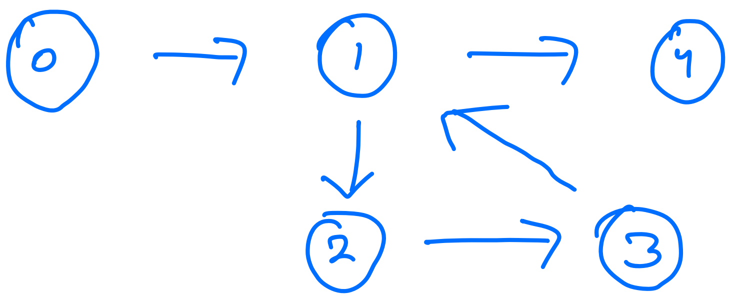
4. 5 6 7.



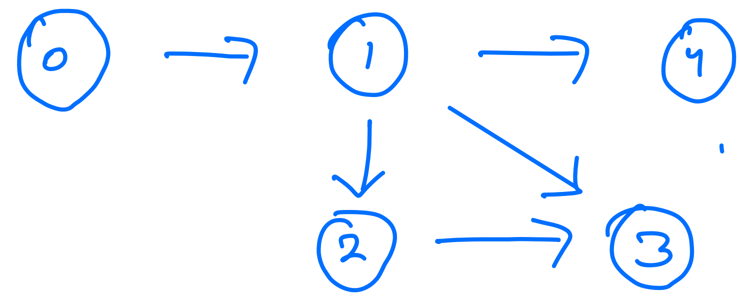
T.C → O(N+E)

S.C → O(N) + O(N) (Recursive stack)

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



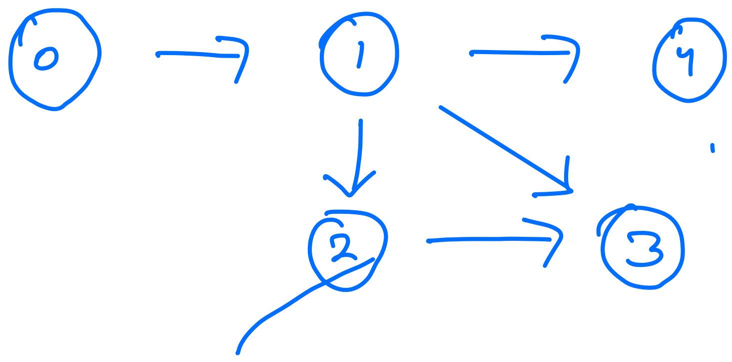
true.

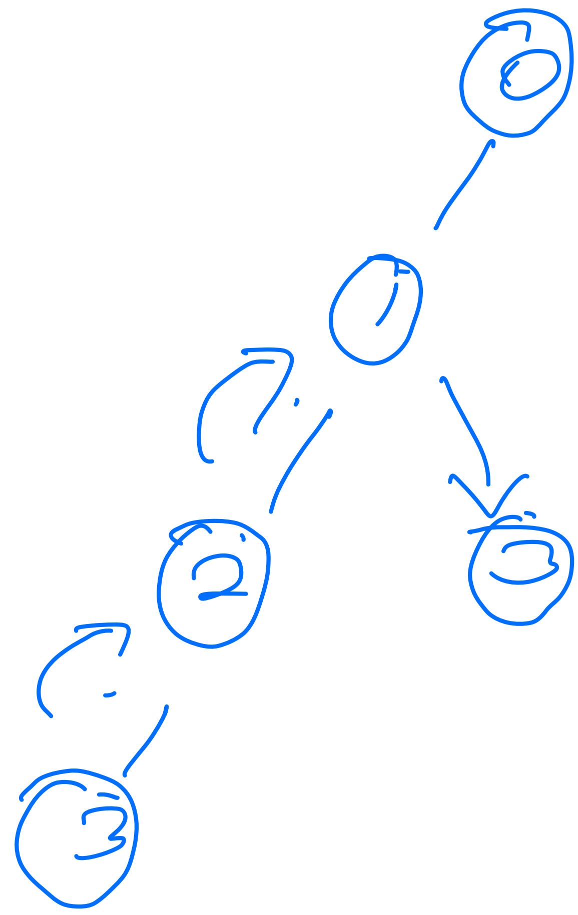


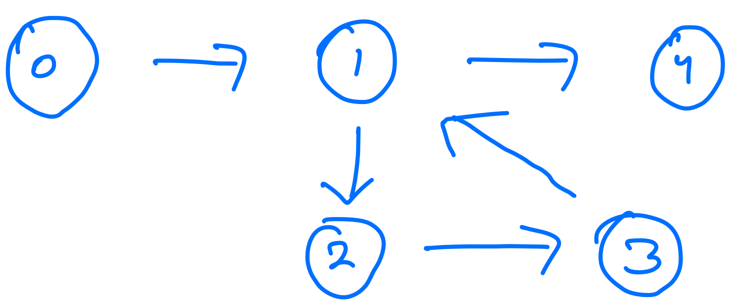
false.

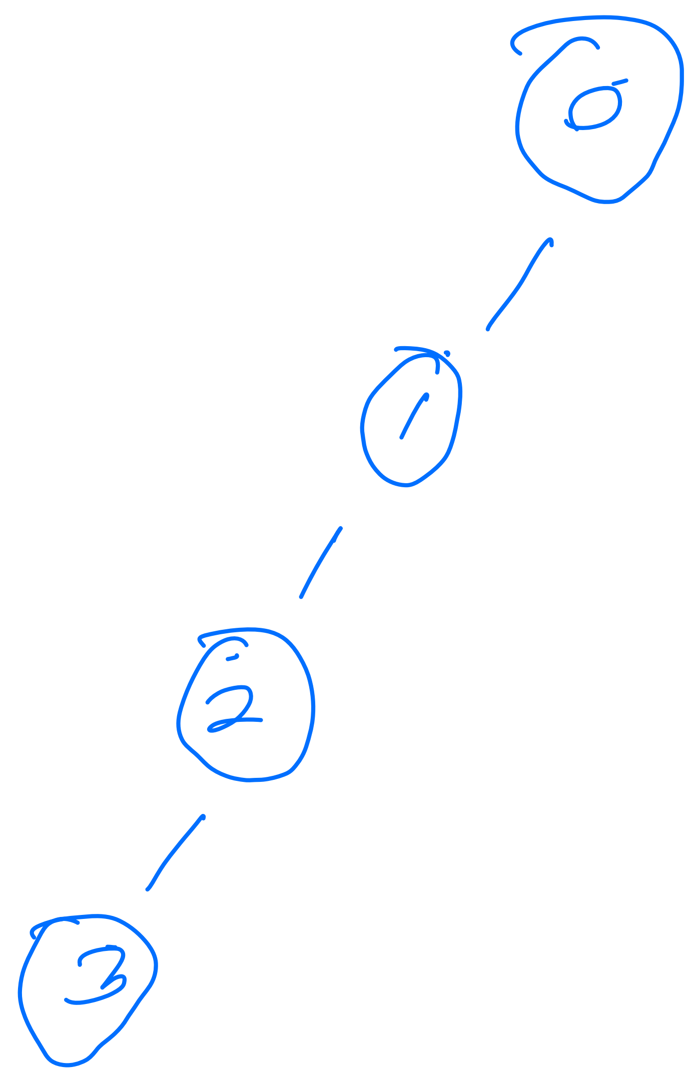
→ If a visited node is encountered again during my traversal → Cycle exists. ✗

→ If a visited node in current path is encountered again → a cycle exists. ✓









Code:

boolean visited [N]; $\forall i$ vis[i] = f;

boolean path [N]; $\forall i$ path[i] = f;

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


boolean   dfs( graph, src, visited, path) {
        visited [src] = true;
        path [src] = true;

    for ( int  nbr :  graphs [src]) {
        if ( path [nbr] == true) {
            return  true;
        }

        if ( visited [nbr] == false
            && dfs (graph, nbr, vis, path))
            { return  true; }.
    }
    path [src] = false;
    return false;
}
```

T.C → O(N+E)

S.C → O(N).