- What are Graphs?
- Graph Representation
- BFS
- DFS
- Find cycle in detected graph

Mock Interview : 60 mins

a) Intro and Flow Explanation by Mentor (3-5 min)

b) Q and A (25 mins)

c) Feedback and Discussion (5 mins)

Ps → sat 27 Jan

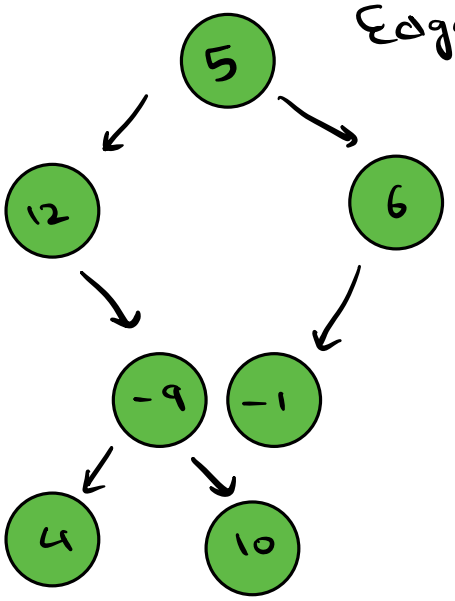
4-5 Qus

Introduction to Graphs

Graph - A collection of nodes connected to each other using edges.

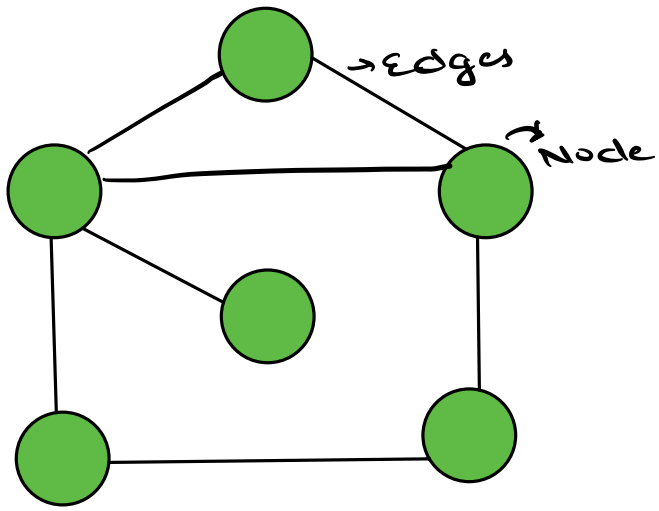
Tree

N = 7
Edges = 6



Graph

N = 7
Edges = 7



Tree is a Directed Graph

1. Tree is a hierarchical data structure
2. N nodes → N-1 Edges

Instagram

A → B

Directed

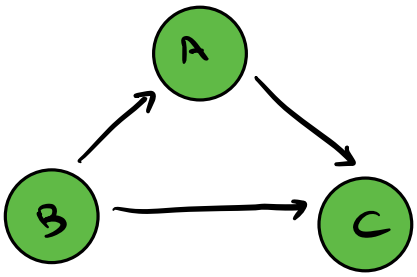
Facebook

A — B

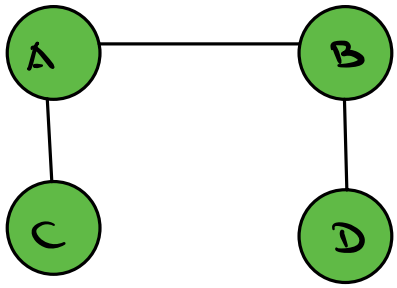
Undirected

Classification of Graphs

I

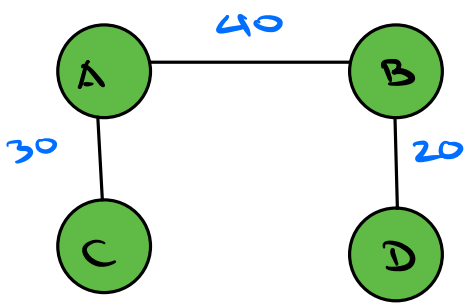


Directed Graph

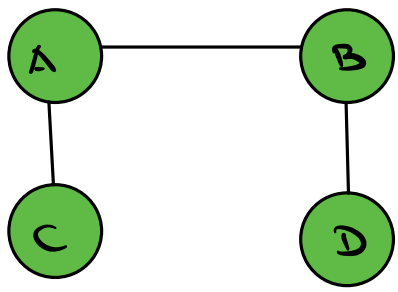


Undirected Graph

II

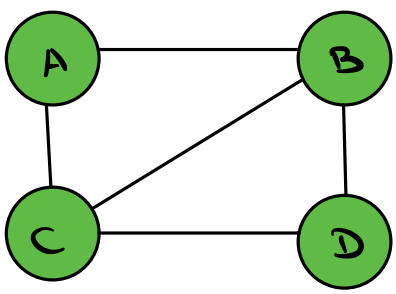


Weighted graph

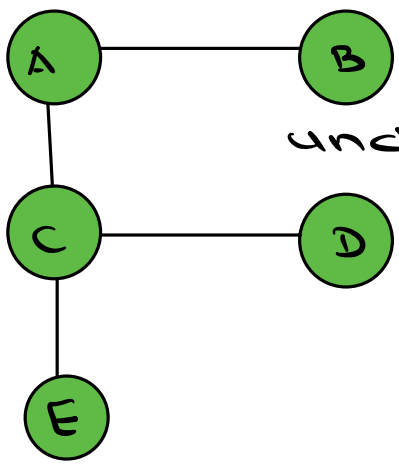


Unweighted Graph

III

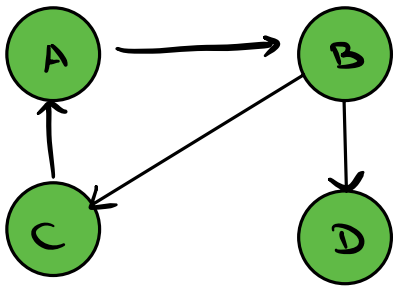


Undirected Cyclic Graph

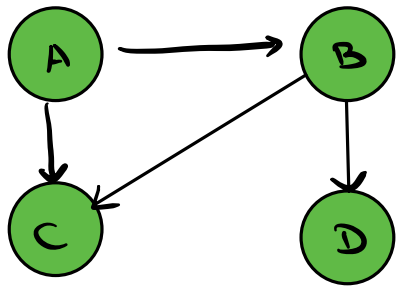


Undirected Acyclic Graph

Directed Cyclic Graph



Directed Acyclic graph



Cyclic Graph -

Start from a node and you reach same node back without repeating an edge

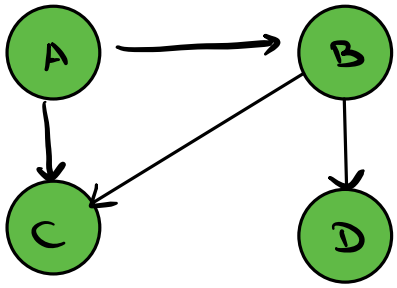
Indegree of a Node

No. of edges which are coming into that node

indegree (C) = 2

Outdegree → outgoing edges

Outdegree (C) = 0
Outdegree (A) = 2



Simple Graph -

A graph in which there is no self loop and no multiple edges b/w any 2 nodes

What Input is given for Graph?

(Given an undirected graph with N nodes & M edges)

1st line → N M

M lines where each line contains u v

↓

edge b/w u and v

N M

6 6

2 3 ✓

4 5 ✓

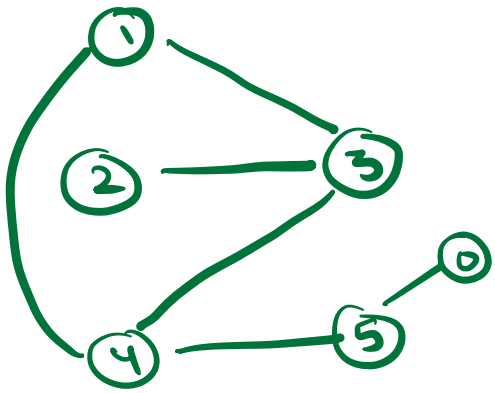
0 5 ✓

4 1 ✓

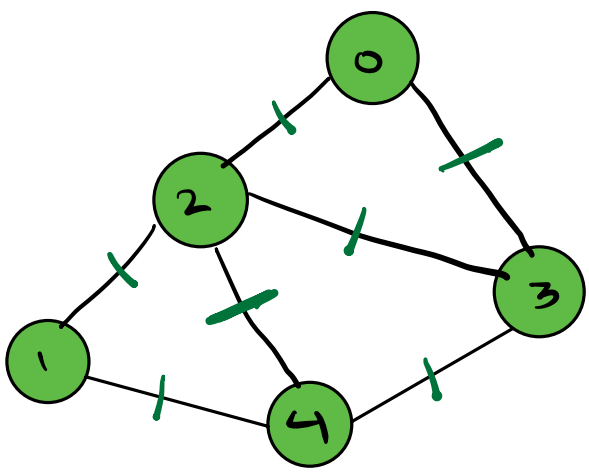
1 3 ✓

3 4 ✓

6 → 1 to 6 OR 0 to 5



Storing a Graph



Input

5 7

1 4 ✓

2 0 ✓

3 2 ✓

4 3 ✓

2 4 ✓

3 0 ✓

1 2 ✓

2 → 4

1 ⇄ 4

Representation 1 (Adj Matrix)

int mat[N][N] → mat[5][5]

u → v

cols → neighbours

rows → Actual nodes	0	1	2	3	4
0	0	0	1	1	0
1	0	0	1	0	1
2	1	1	0	1	1
3	1	0	1	0	1
4	0	1	1	1	0

0 → No edge

1 → edge

Given N and E, mat[N][N]

// u, v

	Unweighted	Weighted
Undirected	mat[u][v] = 1 mat[v][u] = 1	mat[u][v] = w mat[v][u] = w
Directed	mat[u][v] = 1 u → v	mat[u][v] = w

// u v w

6 7

u v w

5 Nodes → 100 cells

5 edges → 10 cells

N^2 cells

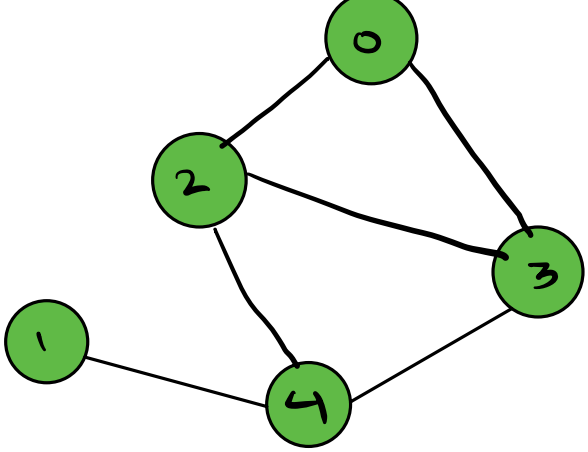
Space Wastage

Storing a Graph

S (List)

↓

N + E



Input

5 6

1 4

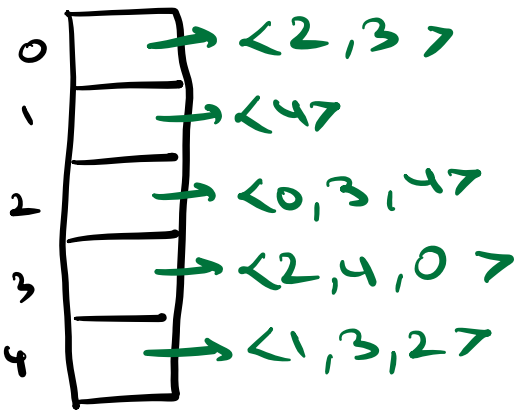
2 0

3 2

4 3

2 4

3 0



list<int> ar[5]

list<int> ar [N]

Representation 2 (Adjacency List)

// u v w

// u,v

	unweighted	weighted
Undirected	ar[u].add(v) ar[v].add(u)	ar[u].add(<v,wt>) ar[v].add(<u,wt>)
Directed	ar[u].add(v)	ar[u].add(<v,wt>)

u → v

10:22

list<pair> ar[N]

↓

pair <int,int>

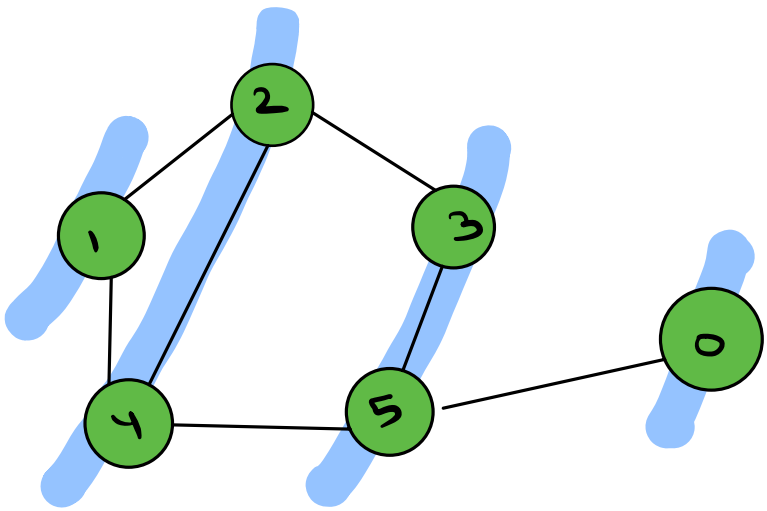
nbr wt

Breadth first search → Level order Traversal

1. Given an undirected graph, source node and dest node. Check if node can be visited from source node.

Input

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

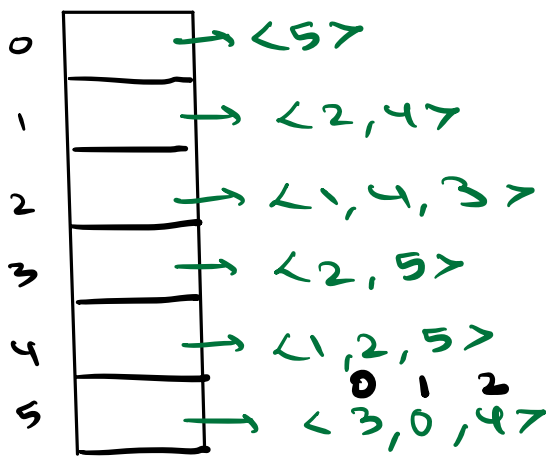


Src	Dest
1	0

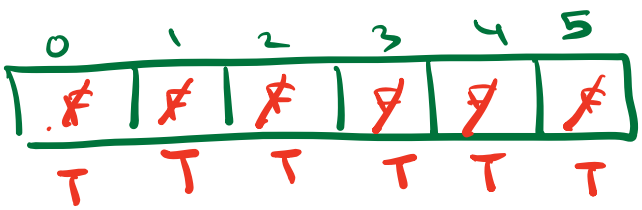
True

Queue

1 2 4 3 5 0



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



visited [n]

↓

6

If visited [dest] = T

path exists

else

path does not exist

```
bool bfs ( list<int> adj[] , int s , int d ){

    bool vis[N] = <F>          //Nodes N
                                [0 - N-1]

    Queue<int> q
     q.enqueue(s)
     vis[s] = true

     while ( q.size()>0) <

          int u = q.front()
            q.dequeue()
            print(u)                adj[u]

          //put u's unvisited neighbours

          for(i=0 ; i< adj[u].size() ; i++)<

                nbr = adj[u][i]
                if (vis[nbr] == false) <
                    q.enqueue(nbr)
                    vis[nbr] = true
                >
          >
     >

     return vis[d]

}
```

TC: O(V+E)
SC: O(V)
visited[]
queue()

V → Nodes
E → Edges

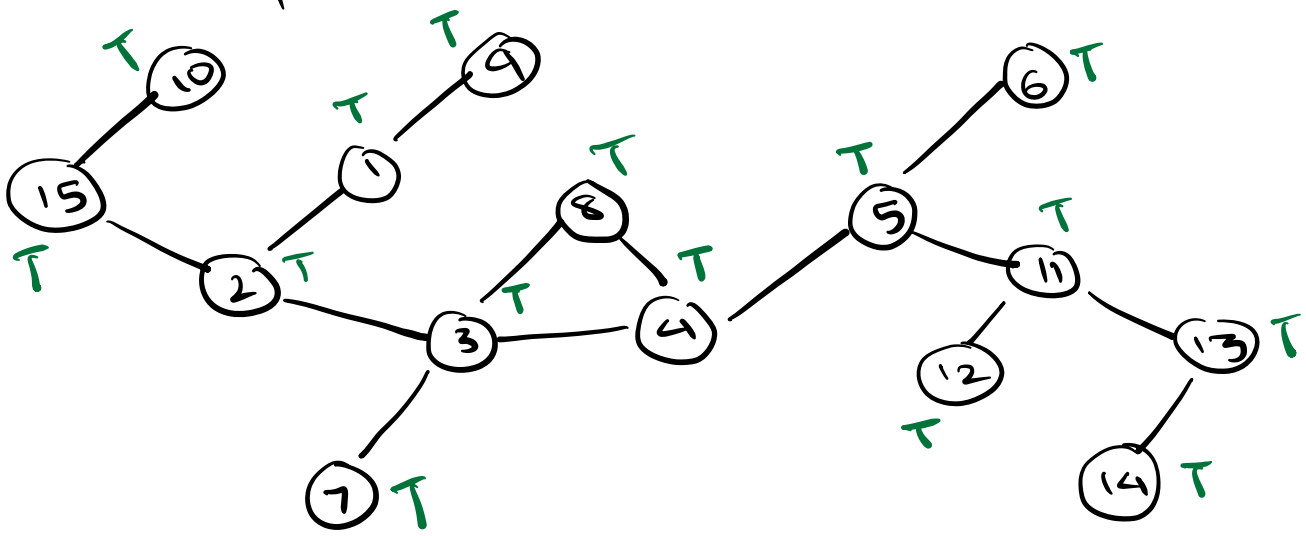
```
for (i=1 ; i<=n ; i++){
    for (j=1 ; j <= n ; j++) {
    }
}
```

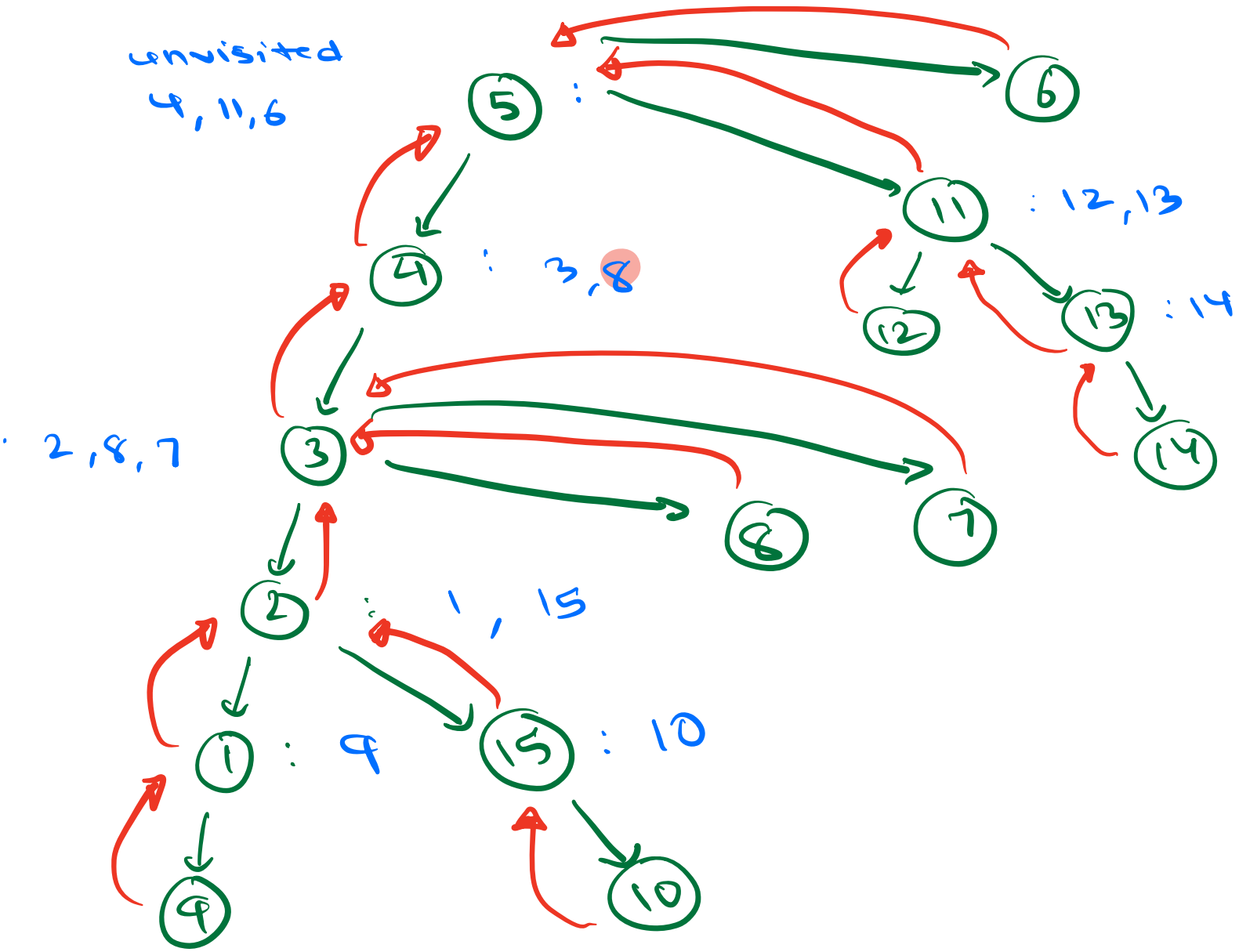
i	j	Total
1	1 → n	n
2		n
⋮		⋮
n		n
n		n^2

Total = Outer + Inner

$= n + n^2$

DFS (Depth First Search)





DFS(src, adj, vis)

```
void DFS(int u, list<int> adj[], int vis[]){
    vis[u] = true
    for (i=0 ; i < adj[u].size() ; i++){
        nbr = adj[u][i]
        if (vis[nbr] == false)
            dfs(nbr, adj, vis)
    }
}
return vis[d]
```

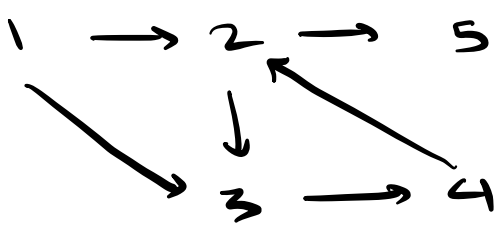
$V + E$ /

TC : $O(N + E)$

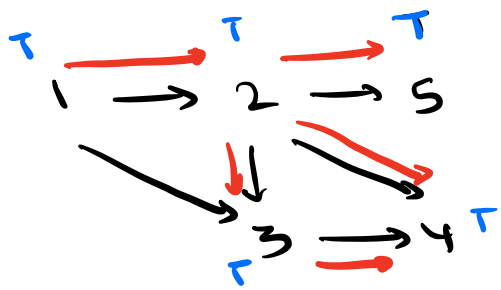
SC : $O(N)$

List → $N + E$ / $V + E$

3. Check if a simple directed graph has a cycle



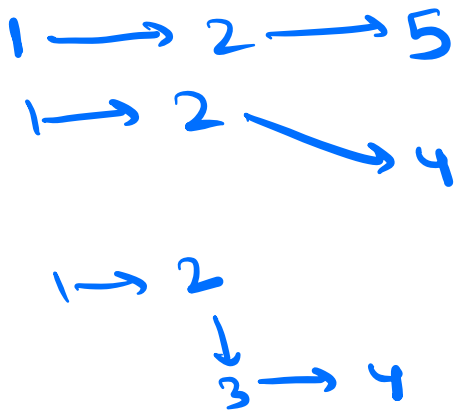
True



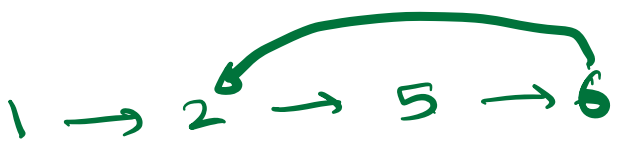
False

If a visited node is encountered again → cycle ✗

A node can be a part of multiple paths



If a node is encountered again in current path → cycle

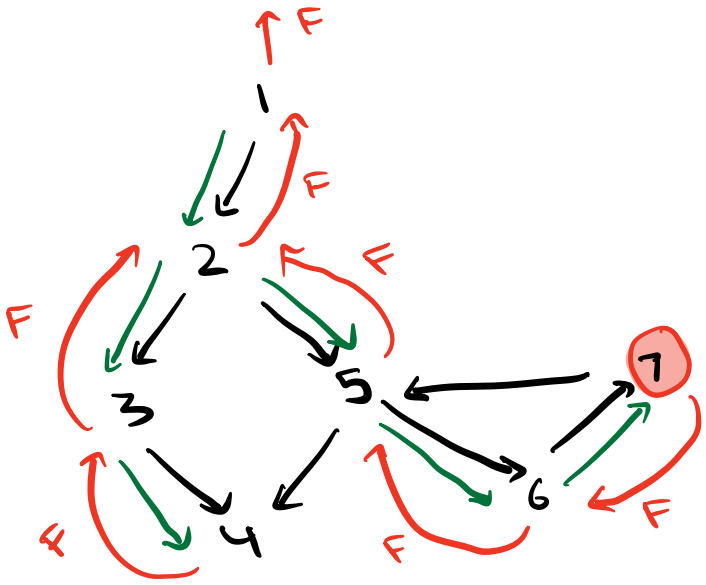


YES

```
list<int> path
bool cycle(int u, list<int> adj[], int vis[]){
    vis[u] = true
    path.add(u)

    for (i=0 ; i < adj[u].size() ; i++){
        nbr = adj[u][i]
        if (path.contains(nbr)
            return true

        else if (!vis[nbr]){
            if (cycle(nbr, adj, vis)==true)
                return true
        }
    }
    path.delete(u)
    return false
}
```



path (set)

1	2	5	6	7

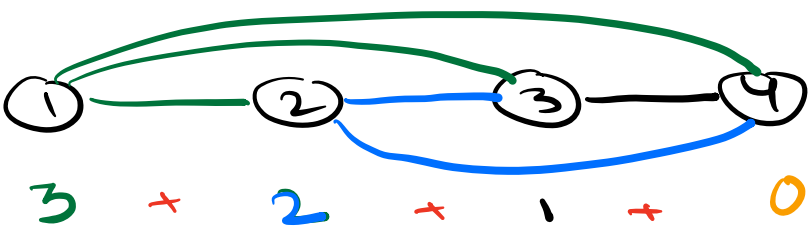
vis

1	2	3	4	5	6	7
T	T	T	T	T	T	T

* Check last page (keep path in a set)

---

4 nody



Max Edgs

6

Min Edgs

3

N Nody

N-1 N-2 N-3 N-4 0

Min Edgs

N-1

Max Edgs

$\frac{N-1\ (N)}{2}$

* How to store path in cycle Finding Ques?

path.contains(nbr) → path[nbr]==T

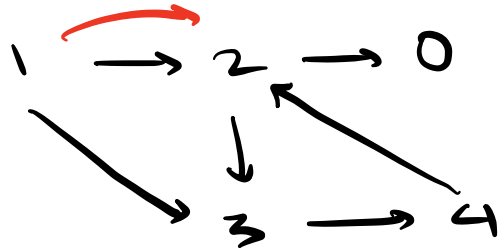
① path → use a hashset (finding any node in hashset → O(1))

② bool path[N]

Whatever comes in the path, mark T

On returning from a node, mark F

N = 5 [0→4]



path[5]