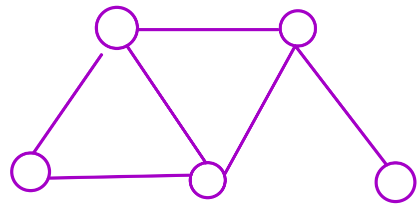
# Graphs 1 : Intro , DFS & Cycle Detection

Graph → collection of nodes & edges.



5 nodes
&
6 edges

→ Google maps

→ Internet

→ Social Media

## Properties

1. Directed (Digraph)

1 → 2

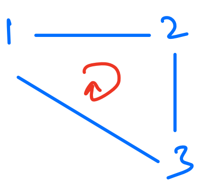
(one-way traversal)

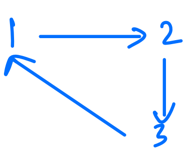
Undirected

1 — 2

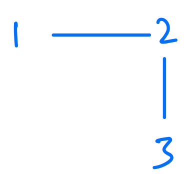
(both way traversal)

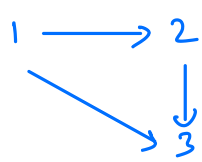
2. Cyclic



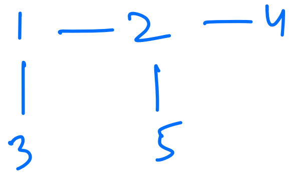


Acyclic

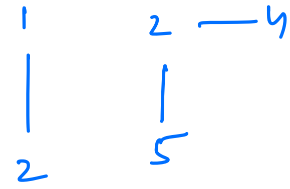




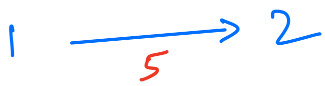
## 3. Connected




## Dis-connected

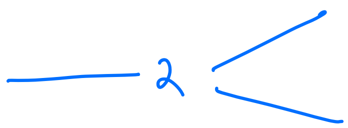



## 4. Weighted



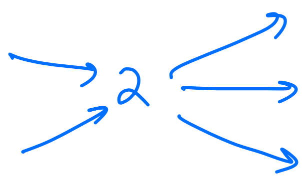

## Un-weighted


## 5. Degree




degree(2) = 3
total connected edges

## Indegree/Outdegree

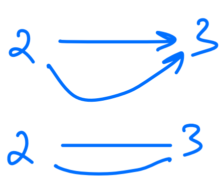



in(2) = 2
total incoming edges

out(2) = 3
total outgoing edges

## 6. Simple graph

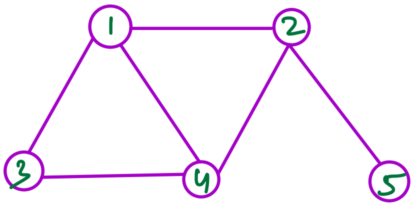
A connected graph without self-loops & multi-edges.



not multi edges

# How to store graph




## 1. Adjacency Matrix

$mat[i][j] = 1$ , $i \to j$

$= 0$ , else

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 1 |
| 3 | 1 | 0 | 0 | 1 | 0 |
| 4 | 1 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 0 | 0 |

mat

// input → A[n][2]

A[i][0] & A[i][1] represent the edge

V → node/vertex

E → edge

```
int mat[N][N] = {0}

for (i=0; i< A.size(); i++){
    u= A[i][0]
    v= A[i][1]

    mat[u][v] = 1;
    mat[v][u] = 1; //not needed for Digraph
```

A=[ [1,2] ,[1,3] , [2,3] ,[1,5],.... ]

}

$SC = O(V^2)$ or $O(N^2)$

**Advantage** : Easy to update edges

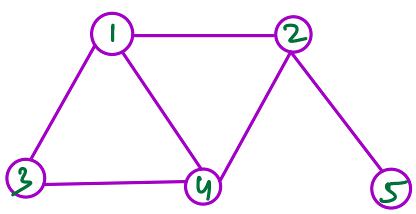
**Disadvantage** : Space wastage if edges are less

if $V = 10^5$ $SC = O(V^2) = O(10^{10})$

not possible to store graph

## 2. Adjacency List

graph[i] → list of nodes connected to i.




graph[1] → {2, 3, 4}

graph[2] → {1, 4, 5}

graph[3] → {1, 4}

graph[4] → {1, 2, 3}

graph[5] → {2}

```
list<int> graph[V];

for(i=0; i<A.size(); i++){
    u = A[i][0]
    v = A[i][1]
    graph[u].add(v);
    graph[v].add(u);
}
```

#nodes #edges

$SC = O(V + E)$

$V <= 10^5$
$E <= 10^5$ } store in adjacency list ✓

if each node is connected to every other node ⇒
each node will have V-1 edges.

total edges = $V*(V-1)$
$= O(V^2)$

$E = O(V^2)$

## for weighted graph - how to store

### Adjacency Matrix

$$mat[i][j] = \cancel{1}\, W \quad , \quad i \xrightarrow{w} j$$
$$= 0 \quad , \quad else$$

### Adjacency List

graph[i] = list of (node, weight) connected to i

$$\begin{tikzcd}
1 \arrow[r, "5"] \arrow[d, "2"'] & 2 \\
3 &
\end{tikzcd}$$

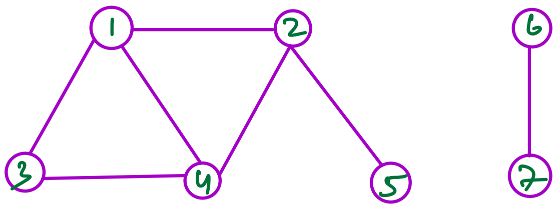
$$graph[1] = \{ (2,5), (3,2), \ldots \}$$

# Graph traversals

pre order
inorder
post order } dfs

level order → bfs

## 1. Depth first Search (DFS)




1 → 2, 3, 4

2 → 1, 4, 5

3 → 1, 4

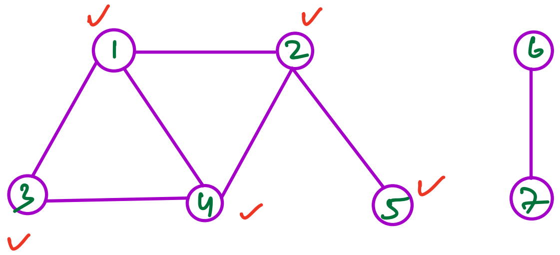
4 → 1, 2, 3

5 → 2

6 → 7

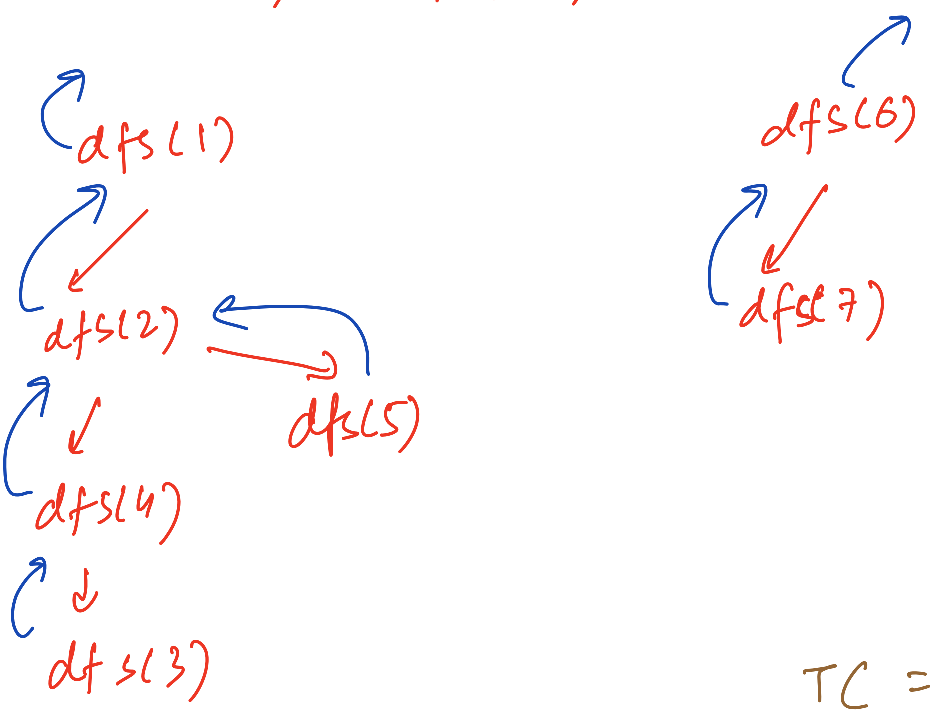
7 → 6

```
visit[V]
∀i, visit[i] = false

for(i=0 to V-1){          → O(V)
    if( visit[i] == false)
        dfs(i)
}

void dfs(u){
    visit[u] = true;
    print(u)
    for( int v : graph[u]){
        if(!visit[v])
            dfs(v)
    }
}
```




| | | | | | | |
|---|---|---|---|---|---|---|
| T ~~F~~ | ~~F~~ T | ~~F~~ T | ~~F~~ T | ~~F~~ T | ~~F~~ T | ~~F~~ T |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

o/p → 1, 2, 4, 3, 5, 6, 7



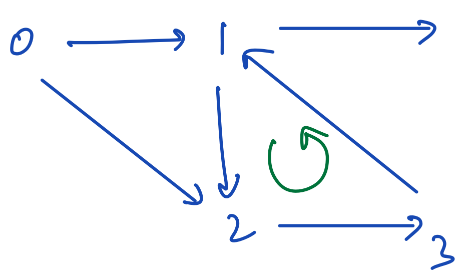

$TC = O(V + E)$

$SC = O(V)$

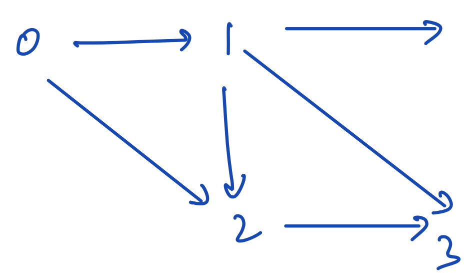
↑ visit array + recursion stack

## Question

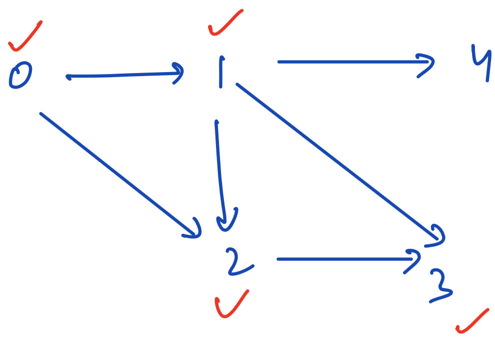
Check if the directed simple graph has a cycle.



ans = true



ans = false



dfs(0)

↓

dfs(1) → cycle is detected since 3 is already visited which is wrong so we can't use visit array to detect cycle.

↓

dfs(2)

↓

dfs(3)

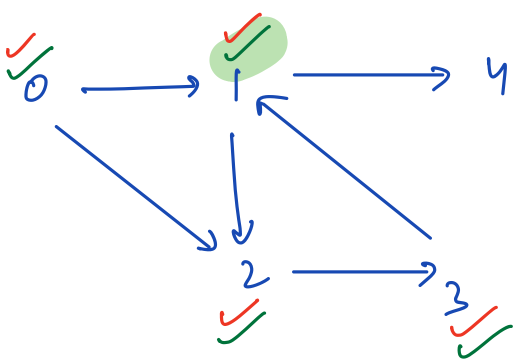
**Solution**: cycle is present if we are visiting a node again which already in current path.

```
visit[V] , path[v]
∀i, visit[i] = false
∀i, path[i] = false
for(i=0 to V-1){  → O(V)
    if( visit[i] == false) {
        if( dfs(i) == true ){
            print(" cycle is detected");
            break;
        }
    }
}

bool dfs(u) {
    visit[u] = true;
    path[u] = true;
    for( int v : graph[u]) {
        if( path[v] == true )
            return true;
        if(!visit[v]) {
            if(dfs(v) == true ) {
                return true;
            }
        }
    }
    path[u] = false;
    return false
}
```

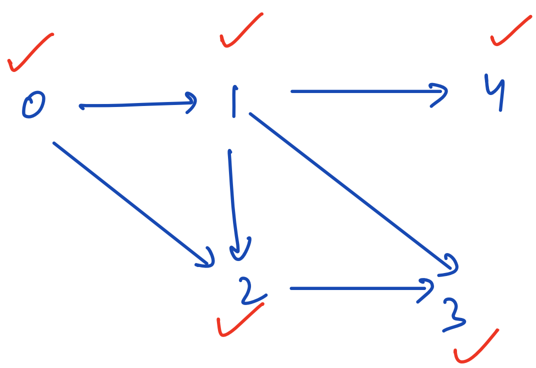
$TC = O(V+E)$

$SC = O(V)$

✓ → visit
✓ → path




0 → 1 , 2
1 → 2 , 4
2 → 3
3 → 1
4 →




NO CYCLE

0 → 1 , 2
1 → 2 , 4 , 3
2 → 3
3 →
4 →