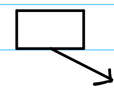
15/9/2023

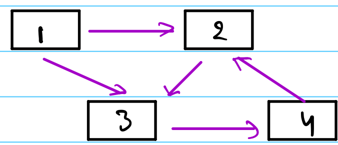
# Graphs - 1

Graph → is a collection of nodes & edges.



node

edge

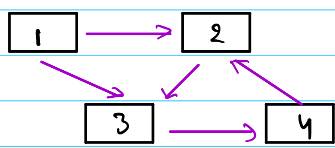


Edges → 5

Nodes → 4

## How graph is stored

1> Adjacency Matrix



$$A = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 1 \\ 4 & 0 & 1 & 0 & 0 \end{array}$$

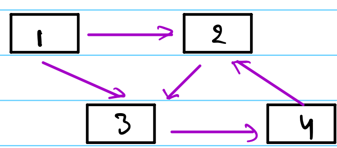
SC: $O(N^2)$

$A[i][j] \to 0$, no edge

$\to 1$ $\boxed{i} \to \boxed{j}$

2> Adjacency List

Adj[i] → list of nodes i is pointing to.



| index | |
|---|---|
| 1 → | {2, 3} |
| 2 → | {3} |
| 3 → | {4} |
| 4 → | {2} |

→ List of list

→ Array of list

→ Map

SC: $O(N+E)$

$N = 10^5$ & $E = 10^5$ → list is much better.

# Properties of graph

1> **Directed**

$\boxed{1} \rightarrow \boxed{2}$

travel only from 1 to 2

**Undirected**

$\boxed{1} — \boxed{2}$

Same as

$\boxed{1} \rightleftarrows \boxed{2}$

travel in both directions

2> **Connected**

1 — 2 — 4
    |
    3 — 5

**Disconnected**

1 — 2 — 4

3 — 5

2 connected components

3> **Weighted**

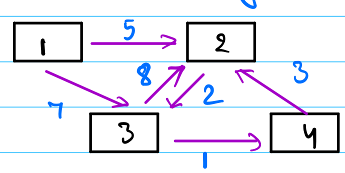
$\boxed{1} \xrightarrow{5} \boxed{2}$

**Unweighted**

$\boxed{1} \rightarrow \boxed{2}$

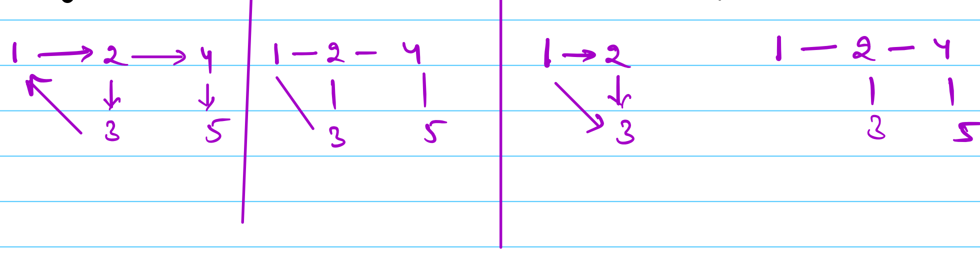
A[i][j] → 0 , no edge
A[i][j] → 5 $\boxed{i} \xrightarrow{5} \boxed{j}$

A[i][j] → weight over the edge i → j



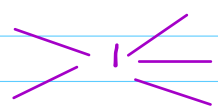

index
1 → { (2,5) , (3,7) }
2 → { (3,2) }
3 → { (4,1), (2,8) }
4 → { (2,3) }

4> **Cyclic** | **Acyclic**



Undirected → cycle of min 3 nodes will be considered.

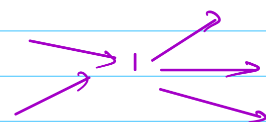
5> **Degree**



d[1] = 5

# connected edges of a node is degree of that node

**In degree / out degree**



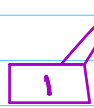
in[1] = 2 out(1) = 3

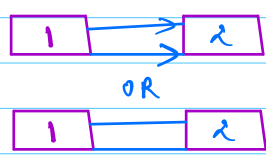
indegree → # incoming edges

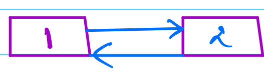
outdegree → # outgoing edges.

6> **Simple graph** → connected graph without self loops & multiedges

or



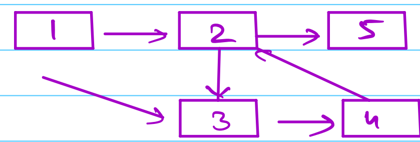


→ Not multiedge

# Traversal

1> Depth first search (DFS)




Go deep till it is possible. Once a path is complete, backtrack to try alternate paths

1> Travel all nodes only once.

2> Keep a track of visited nodes

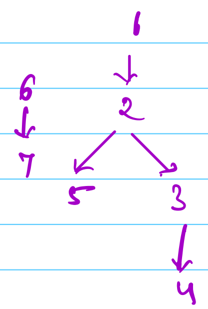
3> Check if all nodes are travelled before exit.

$\forall i$, vst[i] = false

```
for i -> 1 to N {
    if (!vst[i]) dfs(i);
}

void dfs(u) {
    vst[u] = true
    print(u);

    for (v : Adj[u]) {
        if (!vst[v]) {
            dfs(v);
        }
    }
}
```



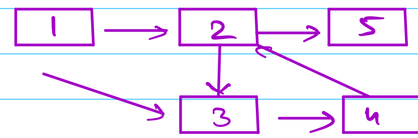

1 → {2}
2 → {3, 5}
3 → {4}
4 → { }
5 → { }

o/p → 1 2 3 4 5 6 7

$SC: O(\underbrace{N}_{vst[]} + \underbrace{N}_{recursion}) \simeq O(N)$

$TC: O(N+E)$

## 2> Breadth first search (BFS)




∀i, vst[i] = false

```
for i→1 to N {
    if (!vst[i]) bfs(i);
}
```

1> Travel all nodes only once.

2> Keep a track of visited nodes

3> Check if all nodes are travelled before exit.

```
void bfs(u){
    vst[u] = true;
    q.enqueue(u);
    while(!q.isEmpty()){
        x = q.dequeue()
        print(x);
        for (v: Adj[x]) {
            if (!vst[v]) {
                vst[v] = true;
                q.enqueue(v);
            }
        }
    }
}
```

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

o/p → 1 2 3 5 4

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

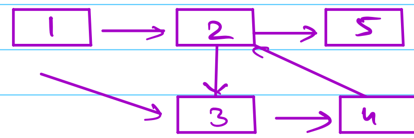
1 2 3 4 5

SC: $O(\underset{vst}{N} + \underset{Q}{N})$

TC: $O(N+E)$

Meet at 8:25 am IST

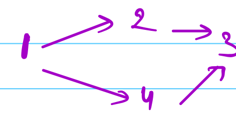
Q Check if the simple directed graph has a cycle.



If a visited node is travelled again ⇒ cycle ✗

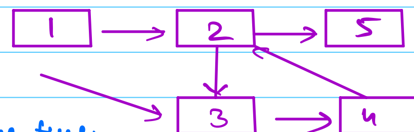
Ans = true

If a visited node in same path is travelled again ⇒ cycle



travel a path ⇒ DFS

∀i, vst[i] = false



```
for i = 1 to N {
    if (!vst[i] && dfs(i)) return true;
}
```

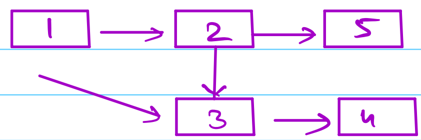
1 → {2, 3}
2 → {3, 5}
3 → {4}
4 → {2}
5 → { }

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

    for (v : Adj[u]) {
        if (path(v)) return true;

        if (!vst[v] && dfs(v)) {
            return true;
        }
    }
    path[u] = false;
    return false;
}
```

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| vst | ✓ | ✓ | ✓ | ✓ | |
| path | ✓ | ✓ | ✓ | ✓ | |




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| vst | ✓ | ✓ | ✓ | ✓ | ✓ |
| path | ✓ | ✗ | ✗ | ✓ | ✓ |

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

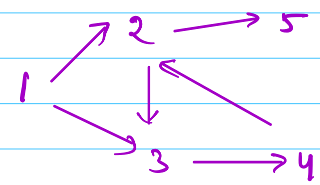
TC: O(N+E)
SC: O(N)

Q Given N courses with pre-requisite of each course.
Check if it is possible to complete all courses.

i/p → X is a prerequisite of . . . . . Adj list

1 → { 2, 3 }

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




cyclic graph → false

acyclic graph → true.

Course schedule - I
Course schedule → <u>II</u>