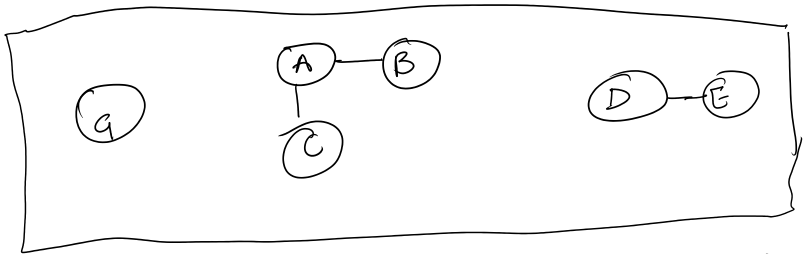
# Graphs

$G = (V, E)$ — $V$: set of nodes / vertices, $E$: set of edges



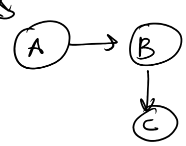

$V = \{A, B, C, D, E, G\}$

$E = \{(A,B), (A,C), (D,E)\}$

$G = (V, E)$

Graphs — Directed - edges are ordered pairs

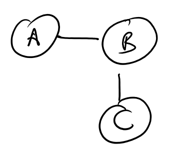
$(A,B) \neq (B,A)$




Graphs — Undirected

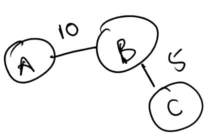
edges are unordered pairs

$(A,B) = (B,A)$

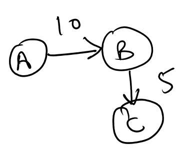



* Weighted graph

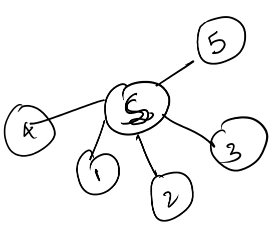
↓ edges has weight



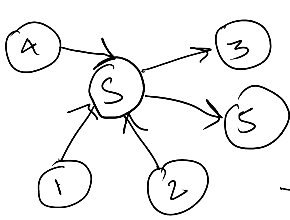

or




* Adjacency. - set of nodes which are adjacent / neighbor of 'S'




$adj[S] = \{1, 2, 3, 4, 5\}$

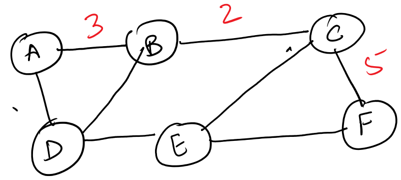



$adj[s] = \{3, 5\}$

→ outgoing edges

* Path - contigous sequence of edges

Eg.




Path from A to F → Length = 3 ↓ non weighted

{ (A, B), (B, C), (C, F) }

Or 10 (3+2+5)

{ (A, D), (D, E), (E, F) }

- Length of the path → # no. of edges

  or total weight of edges

'F' is **reachable** from 'A' if there is a path from A to F

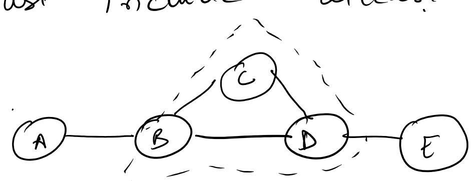
Any two nodes are said to be **connected** if there is a path from one to the other.

* CYCLE - path whose start and the end nodes are the same

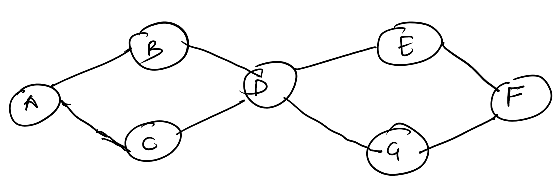
- Simple cycle - if all nodes are distinct except the first & the last

  must include atleast 3 vertices

Eg.




A B D E F G D C A

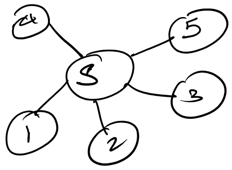


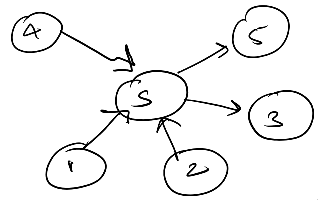

not a simple cycle

- No cycle - Acyclic Graph

Directed graph with no cycle - Directed Acyclic graph (DAG)

* Degree of node - no. of edges connecting the node



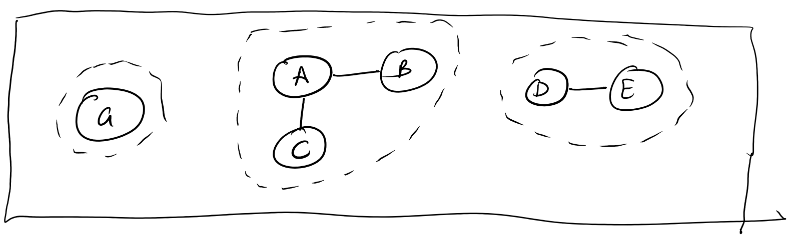




$deg(s) = 5$

$indeg(S) = 3 \rightarrow \{1, 2, 4\}$

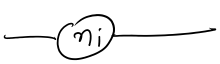
$outdeg(S) = 2 - \{3, 5\}$

* Connected Components → subset of nodes that are connected



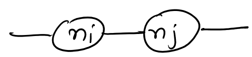

Connectivity - Equivalence relation b/w set of nodes

**Reflexive**




Each node is in a path of length 0 with itself

**Symmetric**




$(n_i, n_j) \in Path$

$\Rightarrow (n_j, n_i) \in Path$

**Transitive**


$(n_i, n_w) \in Path$

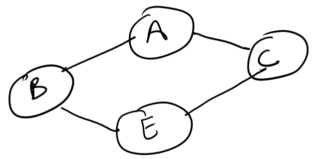
$(n_w, n_j) \in Path$

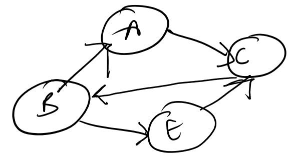
$\Rightarrow (n_i, n_j) \in Path$

Connection — Strongly connected

path from each node to every other node

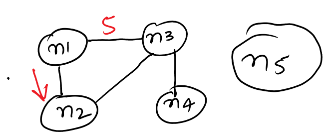
Weakly connected




| | Tree | Graph |
|---|---|---|
| Cycle | ✗ | ✓ |
| Root | ✓ | ✗ |
| Reachability of Nodes | ✓ | ✗ |
| Direction | ✓ | Optional |

## Graph Representation



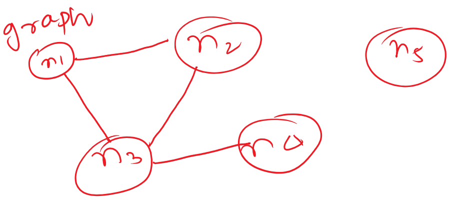

### Adjacency Matrix

| | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ |
|---|---|---|---|---|---|
| $n_1$ | 0 | 1 | ~~1~~ 5 | 0 | 0 |
| $n_2$ | 1 \| 0 | 0 | 1 | 0 | 0 |
| → $n_3$ | ~~1~~ 5 | 1 | 0 | 1 | 0 |
| $n_4$ | 0 | 0 | 1 | 0 | 0 |
| $n_5$ | 0 | 0 | 0 | 0 | 0 |

$O(N^2)$ - Space

Dense graph

Adjacency = O(1)




### Adjacency List

- $n_1$ → $n_2$ → $n_3$ /
- $n_2$ → $n_1$ → $n_3$ /
- $n_3$ → $n_1$ → $n_2$ → $n_4$ /
- $n_4$ → $n_3$ /
- $n_5$

$O(N^2)$ ⇒ worst case

Sparse graph

Adjacency - O(N)

## ✱ Graph Traversal — Breadth First / Depth First

Graph Traversal (G, V) — starting index

{

Put 'V' into Container C

```
{
    Put 'v' into Container C
        While ( container 'c' is not empty)
        {
            Remove 'x' from container C
            if ('x' has not been visited)
            {
                visit 'x'        { x.visited = true }
                for ( each vertex 'w', adj X)
                {
                    if('w' has not been visited)
                        Put 'w' into Container C
                }
    }   }   }
```