GRAPH ALGORITHMS

▷ Notation

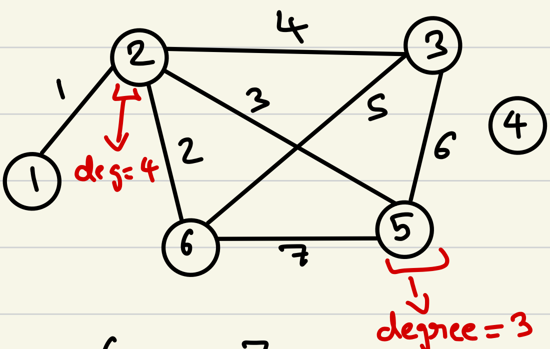
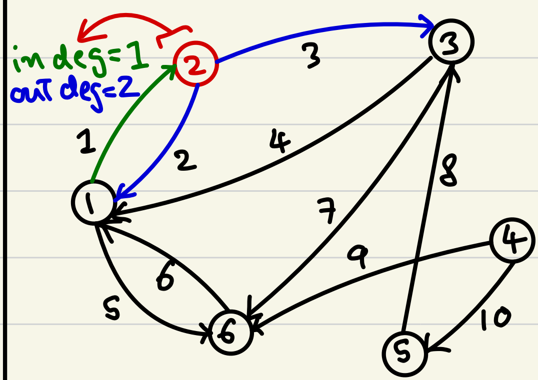
$G = (V, E)$

V: vertices, $|V| = n$

E: edges, $|E| = m$

If $e \in E$, then $e = (u, v)$ s.t. $u, v \in V$

- No multiple edges
- No self-loops.

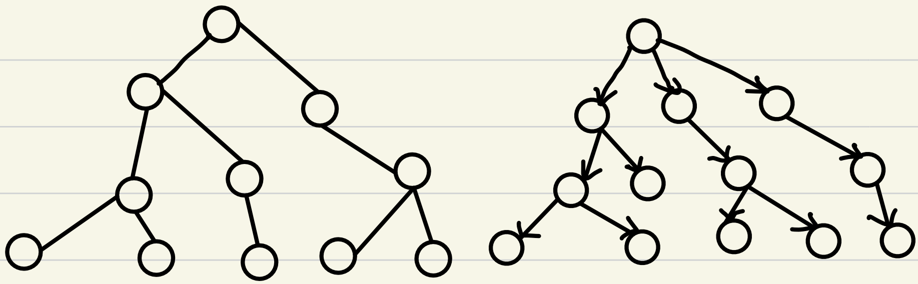
Undirected	Directed
	
$n = 6$, $m = 7$	$n = 6$, $m = 10$
Max edges: ${}^nC_2 = \frac{n(n-1)}{2}$	Max edges: ${}^nC_2 * 2 = n(n-1)$
$\boxed{\sum \deg(v) = 2m}$	$\boxed{\sum \text{in-deg}(v) = \sum \text{out-deg}(v) = m}$

▷ Trees

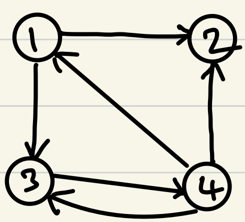
▷ A subset of graphs, s.t.:

- $m = n - 1$
- It's connected
- Has no cycles

→ Any 2 of these conditions are sufficient for a graph to be a tree



▷ Graph Representation



Adjacency Matrix	Adjacency List

Adjacency Matrix

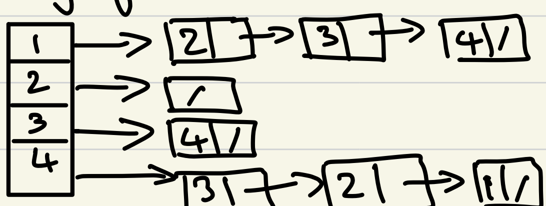
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

▷ Takes n^2 space

▷ Fast access

Adjacency List

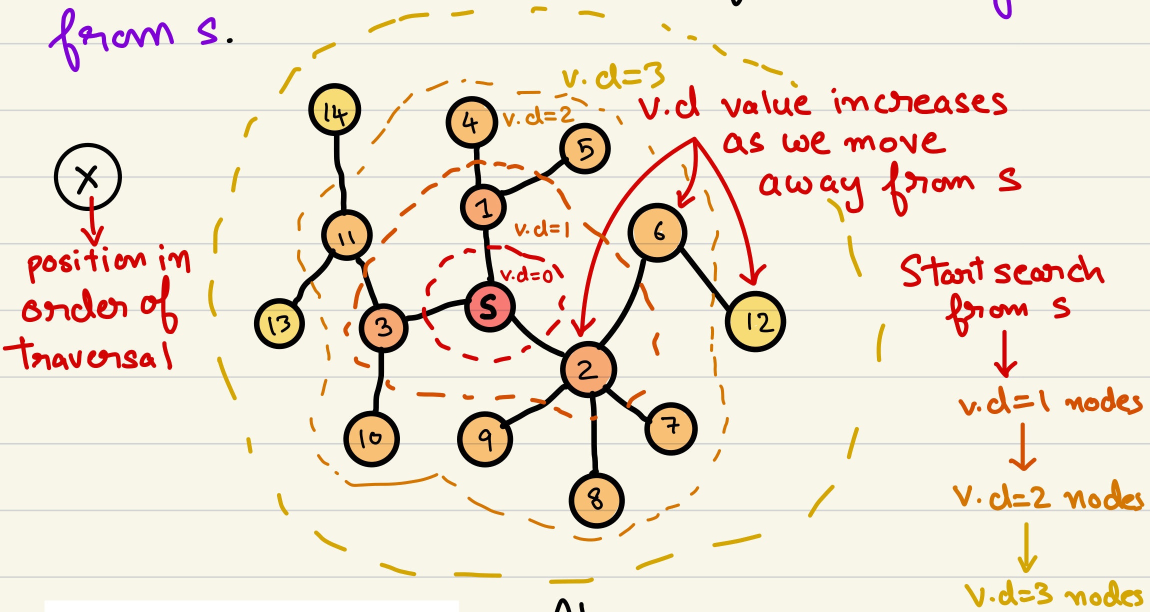
Array of lists



▷ Space efficient, but slow access

▷ Breadth First Search (BFS)

Given graph $G = (V, E)$, $s \in V$, BFS traverses nodes in G in the order of increasing distance from s.




```
BFS(G, s)
    for each vertex u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
    s.color = GRAY
    s.d = 0
    s.π = NIL
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
            if v.color == WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.color = BLACK
```

Algo:

① Add s to queue Q

② Until Q is empty,

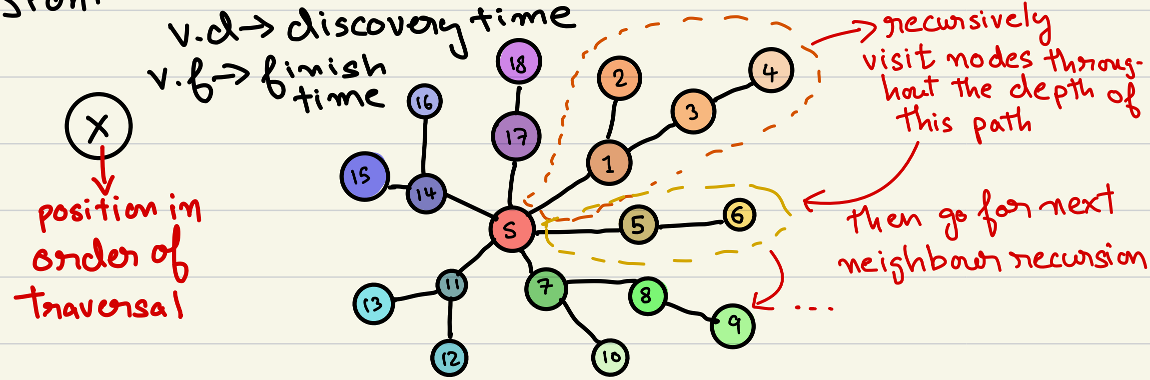
(i) Dequeue from Q to get node v

(ii) Enqueue v's neighbours

(iv) Mark v as visited

▷ Depth First Search (DFS)

DFS traverses nodes in the order of their recursive occurrence starting at node s. In other words, we will recursively visit immediate neighbours, thereby going first for the depth of recursion.




```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1              // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]        // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK              // blacken u; it is finished
time = time + 1
u.f = time
```

Algo: For every node s

1. Start at node s
2. For each un-visited neighbouring node "v", recursively DFS on it.

Paranthesis Theorem

For a DFS traversal of a graph $G(V,E)$, either one of the following holds true for $\forall\, u,v \in V$.

- $u.d < v.d < v.f < u.f$
- $u.d < u.f < v.d < v.f$

White-Path Theorem

In a graph $G(V,E)$, $\forall\, v,u \in V$, v is a descendant of u iff at time $u.d$, $\exists$ "white path"

$$u \overset{}{\underset{path}{\leadsto}} v$$