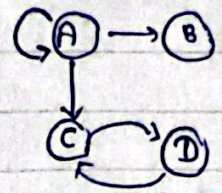
Graphs used to: Wiring electronic components together.
shortest route between 2 cities.
flow of material.
deadlock problems.

Directed graph. / Digraph



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

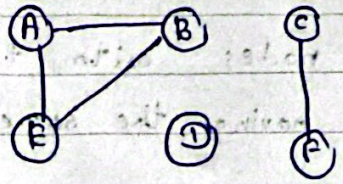
$|E| = 5$

In degree of A = 1
Out degree of A = 3

$V = \{A, B, C, D\}$ $|V| = 4$

$\sum indeg[v] = \sum outdeg[v] = |E|$?

Undirected graph



$V = \{A, B, C, D, E, F\}$ $|V| = 6$

$E = \{\{A,B\}, \{A,E\}, \{B,E\}, \{C,F\}\}$
$\{B,A\}$ $\{E,A\}$

$\sum deg[v] = 2|E|$

$|E| = 4$

degree of B = 2 (no of edges incident on it)

Cyclic - a graph contains a cycle.
Acyclic - a graph not contains a cycle.

Simple path - each vertex is distinct.

* If there is path p from U to V then, V is reachable from U via P.

Complete graph - graph which every pair of vertices is adjacent

→ Undirected → edges = $[V*(V-1)]/2$

→ Directed → edges = $V^2(V-1)$

Connected graph - In a undirected graph, there is no node that is unreachable from any other node.

Strongly Connected - In a directed graph Every node can reach every other node following the direction of edges.

Bipartite graph (V_1, V_2)

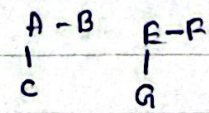
A graph where you can split the vertices into 2 grps so that every edge connects a vertex from V_1 to a vertex from V_2, and no two vertices within the same group are connected.

like, graph where you can color the nodes with two colors without any 2 connected nodes having the same color.

Free tree - connected, acyclic, undirected
doesn't have a root.
every 2 nodes in the tree are connected by exactly one path. (no cycle)

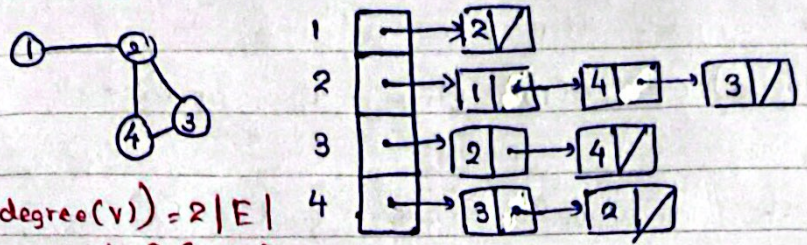
Forest - collection of disconnected trees.
each connected component is itself a tree.
no cycles, not necessarily connected.

ex:-



Rooted tree - one node is chosen as the root.
- parent-child hierarchy.

Adjacency list representation.



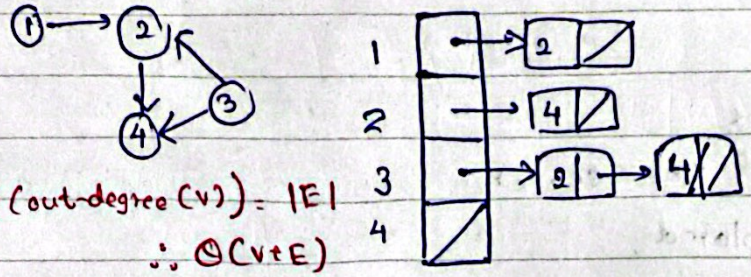
Storage:

nodes = $\sum$(degree(v)) = 2|E|

$\therefore$ $\Theta(V+E)$

** save space for sparse graphs

* order Θ(degree(v))
* must traverse linked list of v.



nodes = $\sum$(out-degree(v)) = |E|

$\therefore$ $\Theta(V+E)$

Sparse graph - few edges compared to the no. of vertices

Dense graph - edges, close to the maximum possible
almost every node is connected to every other node.

Matrix representation

Advantage: saves space on pointers. (in dense, un-weighted graphs)
faster lookup ($\Theta(1)$)

Disadvantage:

waste space for sparse, weighted graphs.
$O(V^2)$ ⟶ for directed.

for a visit we need to go through a row, even if the graph is sparse. ($\Theta(V)$)

storage - undirected → $O(\frac{1}{2}.V^2)$

BFS

Expand a frontier one step at a time.

(FIFO Queue) ← O(1) time to update.

Computes the shortest distance from root to any reachable node.

White - undiscovered
gray - frontier of the search.
Red - fully explored.

Initialization - O(V) = processing nodes:

Visiting each node:

Adjacency list is searched only once.

Undirected → 2|E|

Directed → |E|

⇐ processing edges.

worst case time = O[V+E]

DFS

can be implemented using:

recursion - Implicit stack

Explicit stack - Iterative.

Time complexity - O(V+E) best for sparse graph.

$O(V^2)$ less efficient for large graphs

Space complexity - O(V) - recursive - due to call stack

O(V) - iterative - for stack storage.

Trees

Leaf node - no children

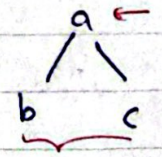
Internal nodes - at least one child

most of the time root is a internal node

Unordered tree.

$$\text{Path} = \binom{\text{nodes}}{\text{involved}} - 1$$

height = 0 (when just the root node)

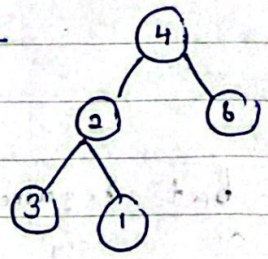


a ← ancestor for b & c

descendents of a

Representation of trees

List



4[2[3][1]][6]

Linked List

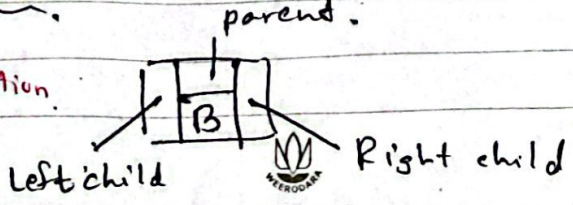
children of each node in a linked list on tree nodes.

in each node points → leftmost child, → right sibling

to represent general trees using a binary tree structure

Linked Structure.

general tree implementation.



Complete binary tree

* every level (except the last) is completely filled.
* last level is filled from left to right.
* every non leaf node has exactly 2 child nodes

Height of a tree $= \log_2 (n+1)$ n = nodes.

Full binary tree

each non leaf node has exactly 2 child nodes.
no node has exactly one child.
when depth = d, no of nodes = $2^{d+1} - 1$

Properties.

if m nodes at level L, then L+1 has at most 2m nodes

Linked representation

data
pointer to the left node.
pointer to the right node.

Sequential representation.

single or one dimensional array. ← but requires a lot of memory space.

root = tree[1]

children of node = tree[k]

tree[2k] tree[2k+1]

array size $= 2^{h+1} - 1$ h = height

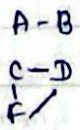
(1) Model Paper.

a) Simple graph
no self-loops
only one edge between any pair of vertices

b) Strongly connected
directed
there is a path from every vertex to every other vertex.

c) Bipartite graph
vertices can be divided into 2 disjoint sets;
all edges connect vertices from one set to the other.
no 2 vertices within the same set are connected.

d) Forest
collection of one or more disjoint trees.
each component is a tree.



e) Complete binary tree
all levels (except the last) are completely filled.
in the last level nodes are filled from left to right with no gaps

f) Full binary tree
every node has either 0 or 2 children.

G) Perfect binary tree
all internal nodes have 2 children
all leaf nodes are at the same level

h) BST

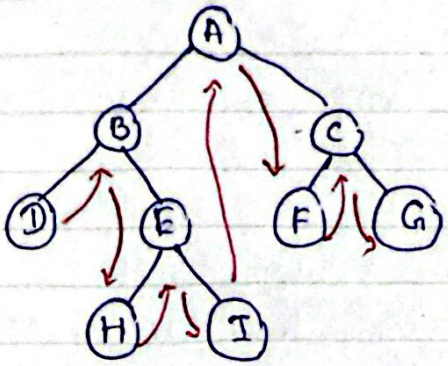
each node has at most 2 children

always → left - less

↳ right - greater

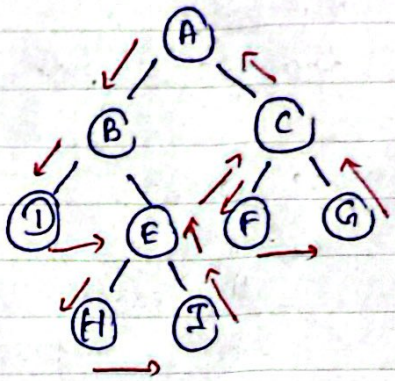
Q2

In-order



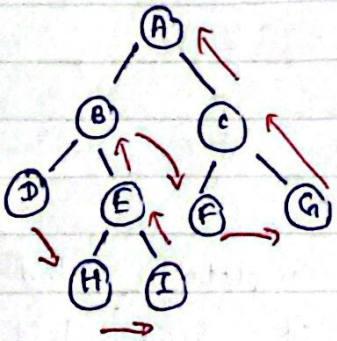
D → B → H → E → I → A → F → C → G

Pre-order



A → B → D → E → H → I → C → F → G

Post-order



D → H → I → E → B → F → G → C → A

~~2023 – DSA I~~

Q2

a) ~~Heap~~

2023 - DSA II

MCQ

17) → Dirac's Theorem

to be a hamiltonian $\begin{cases} n \geq 3 \rightarrow n = \text{vertices} \\ \deg(v) \geq n/2 \quad \text{for every vertex } n \end{cases}$

→ Ore's Theorem

$n \geq 3 \rightarrow n = \text{vertices}$

for every pair of non-adjacent vertices x, y

$\deg(x) + \deg(y) \geq n$

(b), (e) ✓

18) a) ✓

b) ✓

c) X → graph is bipartite if and only if it has no odd-length cycles.

d) X

e) X

31) a)

32) c)

33) a), d)

34) c) a)

35) a) b), c)

2020 DSA II

a) i. j.

$$\begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

Floyd-Warshall Algo

$$\downarrow$$

$$\begin{bmatrix} 0 & 2 & \infty & -4 & \infty \\ \infty & 0 & -2 & 1 & 3 \\ \infty & \infty & 0 & \infty & 1 \\ \infty & \infty & \infty & 0 & 4 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

(02) a) Agree

b) Agree

c) Disagree. → Matrix is better

d) Disagree

BFS → shortest path in an unweighted

DFS → maximum depth or, explore all possible path efficiently.

e) agree

(03) a)

Red-black	AVL
insertion, deletion	Search
large datasets with verifying operations	Height sensitive applications

b) Minimize immediate imbalance.