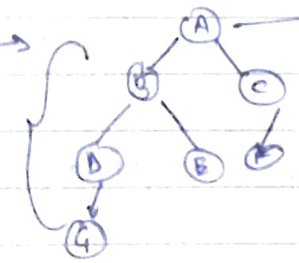
TREES.

(1) Data Structures

- Linear
  - Array
  - Stack
  - Queue
  - LL
- Non Linear
  - Graph
  - Trees → used in opening folders to files/excel/word etc.

(2) Linear only one lvl., Non-L has lvls/hierarchy.

Tree → Graph theory
→ In Data Structure (we are discussing this now)



nodes

→ root of tree.

Can't go from B to A, grows from top to bottom.

A, B, C → Parent node.

D, E, F → child node

(2) Def → Tree can be defined as a collection of nodes which are linked together to stimulate a hierarchy.

Leaf nodes → G, E, F (D is not a leaf node).

Path → ↖ → this is called an edge → collection of edges is called path.

Ancestor → Nodes b/n node & root node connected by path.

Subtree → Node with its decendents.

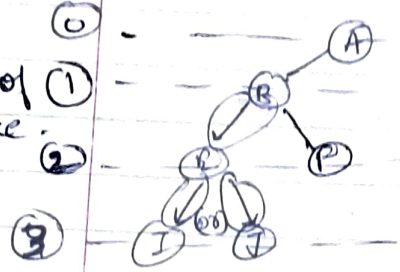
Degree → No: of node (child) of a nodes (parent) Eg:- Degree of A = 3.

Max degree = Degree of this tree.

Depth of node → length of path from root to that node.
i.e No: of edges.

⇒ Depth of root node = 0.

Height of the node → No: of edges in the longest path from that node to the deepest leaf node.



B Height = 2

* Depth starts from root only, not height. Sometimes Depth may be equal to height.

Level of node → Always equal to depth of node

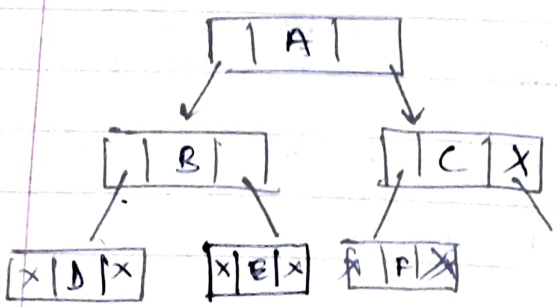
and

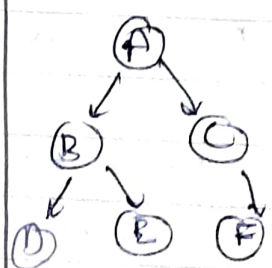
lvl of tree = height/depth of tree.

n nodes = $(n-1)$ edges.

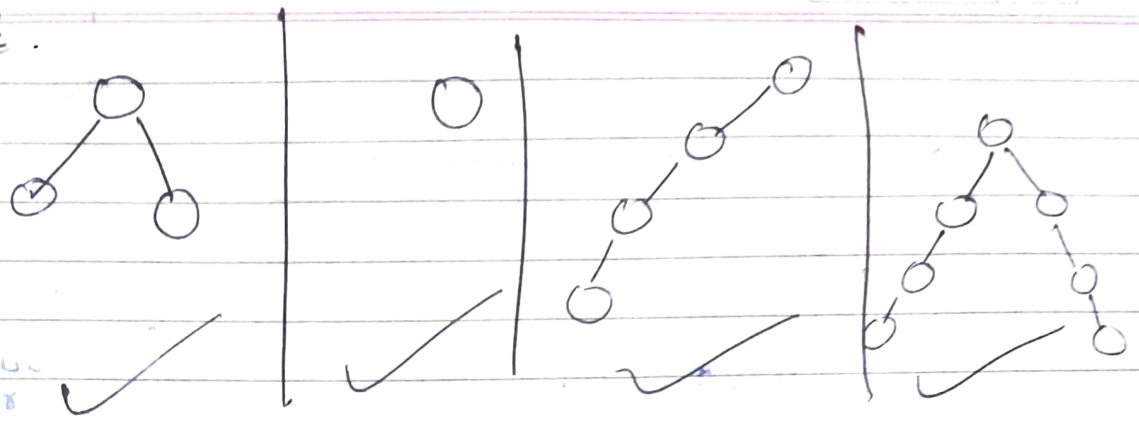
Binary tree

⇒ A node can not have more than 2 children.





Eqs



Props

⇒ Each node has data & may (or) may not have links.
(in middle) (left & right).

⇒ No: of max children/nodes possible in a lvl → $2^{(lvl)}$.

Max no: of nodes for a height h
⇒ $2^{(h+1)} - 1$. — (a)

min ⇒ h+1

Types

- Full/Proper/Strict → 0 (or) 2 nodes for any node (child)
- Complete BTr → (All lvls filled exp last) + (last lvl nodes are in left side)
- Perfect BTr → Every node has 2 nodes except last node.
- Degenerate BTr → All nodes have only 1 child. (can be left or right)

⇒ Every Perf BTr is Compl & Proper BTree.

B Tree	Max node	Min node
	$2^{h+1} - 1$	h+1
full Bin	//	2^{h+1}
Complete Bin Tree	//	2^h

Min height	Max height
$\lceil \log_2(n+1) \rceil - 1$	n − 1
	n − 1/2
//	log n

From (a).

Page No.
Date: YOUVA

Recursion

→ fn calling itself

Implementation of Binary tree.

by LL

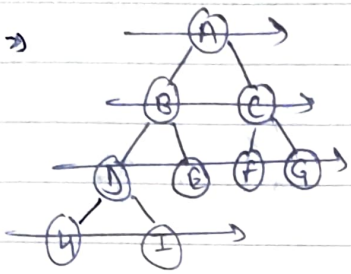
```
Struct node
{ char/int data;
  Struct node *left, *right;
}

Struct node *create()
{ int x;
  struct node *newnode;
  newnode = (structnode *)(malloc)(sizeof(struct node));
  Printf("Enter data");
  Scanf("%d", &x);

if (x == -1)
        { return 0; }

  newnode -> data = x;
Printf("Enter left child of %d", x);
newnode -> left = create();
  Printf("... right ", x);
  newnode -> right = create();
  return newnode;
}
```

```
int main()
{ Struct node *root;
  root = 0;
  root = create();
```

* DS we are creating a copy of newnode created with disturbing the first newnode which still points to root node.

Impl of Binary Tree by Array. (Also called sequential Representation)

★ Used for Heap Sort.

⇒ 

0	1	2	3	4	5	6	7	8
A	B	C	D	E	F	G	H	I

— (a)

If node a ith position/index

then

left child will be at → $[(2*i)+1]$

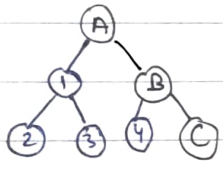
right child will be at → $[(2*i)+2]$

& the node's parent → $\left(\frac{i-1}{2}\right)$

★ To ⇒ represent array the tree should be a complete B Tr.

∴ If H, I present in (E) in above diagram then

7, 8 → [-|-] , 9, 10 = [H|I]



To draw tree A, B, C also.

①, ②, ③, ④ should be considered.

A	-	B	-	-	-	C

∴ Wastage of memory.

Inorder, Preorder & Postorder

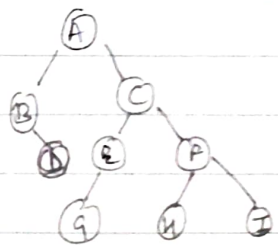
- Left Root Right
- Root Left Right
- Left Right Root.

★ In, Pre & Post take as Root position.

In → B, D, A, G, E, C, H, F, I

Pre → A B D C E G F H I

Post → D, B, G, E, H, I, F, C, A



Coding for Tree Traversals.

⇒ In main :

```
void main()
{ struct node *root;
  Printf("the Preorder is: ");
  Preorder(root);
```

```
Void Preorder (struct node *root)        → can be any name.
{  Printf("%d", root→data);
   Preorder(root→left);
   Preorder(root→right);
}
```

if (root==0) & return 0; }.

recursion used.

Simi for Inorder.

```
void Inorder (st... *root).
{  inorder(root→left);
   Printf("%d", root→data);
   inorder(root→right);
}
```

In Post Order.

```
{  Postorder(root→left);
   Postorder(root→Right);
   Printf(...);
}
```

See drawing trees from just traversals if necessary.