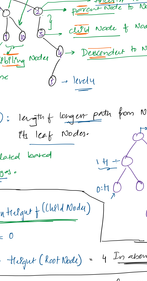
Binary Trees 1
Date: 27/07/2022 Sunday

Hierarchical Data Structure : (Tree)



Leaf Nodes : If it has no child nodes
In that case called leaf Node

Height (Node): length of longest path from Node to any of the leaf Nodes.

Note: Path is calculated based on no. of Edges.

Height (Node)
= 1+ Max Height of (Child Nodes)

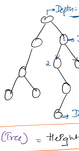
Height (Leaf) = 0

Height (Tree) = Height (Root Node) = 4 In above case

Depth (Node): length of longest path from root Node to Given Node.

→ We don't need this? Because there is only 1 path

Eg:



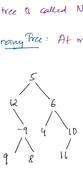
Depth (root) = 0
Depth (Node) = 1+ Depth (Parent Node)

Depth (Tree) = Height (Tree) = Height (Root Node)

→ A tree is called N-Ary Tree: If a node does at most have N child Nodes.

Binary Tree: At most have 2 child Nodes

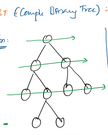
Eg:



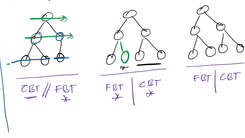
FBT (Full Binary Tree) : Every Node can have 0 or 2 childs

CBT (Complete Binary Tree) : We will fill Nodes level by level from L-R

Eg:



FBT / Not CBT



Height of Tree : 3

Q: min nodes you need to get CBT/FBT of H=3
Max nodes you can have to get CBT/FBT of H=3

	CBT	FBT
min:	8	7
Max:	15	15

FBT H=3 min nodes

H=3 → 7
H=4 → 11
H=5 → 13
H=6 → 13

Generalize : H

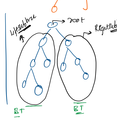
	CBT	FBT
Min:	2^H	$2H+1$
Max:	$2^{H+1}-1$	$2^{H+1}-1$

If In a BT there are N nodes
→ Min Height : $\log(N)$ Height
→ Max Height : $N-1$

N=5 :


```
class Node {
    public:
        int data;
        Node *Left, *Right;
        Node (int x) {
            data = x
            left = NULL
            Right = NULL
        }
}
```

Node to BT



Because of this property of most all the problems can be solved with recursion

```
class Node {
    public:
        int data;
        Node *prev, *next
        Node (int x) {
            data = x
            prev = NULL
            next = NULL
        }
}
```

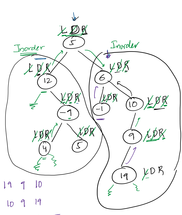
Node to DLL

⇒ Note: Because of similarity of nodes in DLL & BT there are a lot of conversion problems.

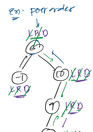
Q: 10 in:

Tree Traversals

1) Pre Order : (D L R)
2) InOrder : (L D R)
3) PostOrder : (L R D)



Eg: Postorder

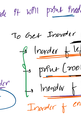


Inorder: 12 4 -9 5 5 -1 6 19 9 10
Preorder: 5 12 -9 4 5 6 -1 10 9 19
Postorder: 4 5 -9 12 -1 19 9 10 6 5

```
void Inorder (Node *head) {
    ① if (head == NULL) return;
    ② Inorder (head → left);
    ③ print (root → data)
    ④ Inorder (head → right);
}
```

Ans: Given root node it will print inorder

Note:

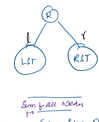


To Get Inorder of entire tree
→ Inorder of left subtree
→ print (root → data)
→ Inorder of Right subtree

Inorder of entire tree

Base Condition: if (head == NULL)

Inorder: 1 2 3 4 (L) (D) (R)
Preorder: 1 3 2 4 (D) (L) (R)
Postorder: 1 2 4 3 (L) (R) (D)



→ Not if nodes

```
int size (Node *head) {
    if (head == NULL) return 0;
    int l = size (head → left)
    int r = size (head → right)
    return l+r+1
}
```

Sum of all Nodes

```
int sum (Node *head) {
    if (head == NULL) return 0
    int l = sum (head → left)
        sum of all LST
    int r = sum (head → right)
    return l+r+root → data
}
```

Max of all Nodes

```
int max (Node *head) {
    if (root == NULL) return 0;
    int l = max (head → left)
    int r = max (head → right)
    return max (l, r, root → data)
}
```

→ return INT_MIN

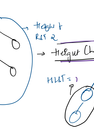
If all data is negative

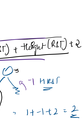
Eg:

Max Val = -1
If no node = 0
Instead of return 0 we will return INT_MIN

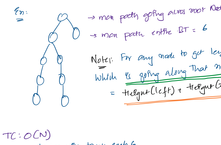
```
int height (Node *head) {
    if (head == NULL) return 0;
    int l = height (head → left)
    int r = height (head → right)
    return max (l, r) + 1
}
```

Eg:





Q8 Given a BT, find length of max path passing through the root Node



→ Height of LST + Height (RST) + 2

Q9 Find max path between any 2 nodes of a BT.

Eg:

→ max path passing through root Node = 5
→ max path in the BT = 6

Note: For every node to get length of max
Which is passing along that node
= Height (left) + Height (Right) + 2

```
Diameter
int ans = 0;
int height (Node *root) {
    if (root == NULL) return -1;
    int l = height (root → left)
    int r = height (root → right)
    l+r+2 // for max path along this node
    ans = max (ans, l+r+2) // Global one variable
                              which we are updating
    return max (l, r) + 1
}
```

→ Diameter of a Tree

→ TC: O(N)
↳ We will touch each of every node

SC: O(H)
↳ Height of tree
↳ recursive call stack.