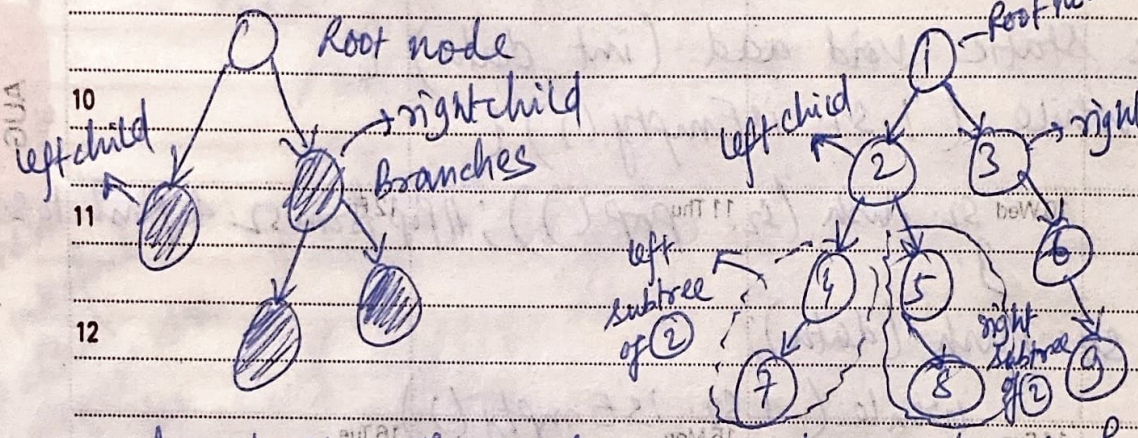
Binary Tree in Data Structure

unlike Stack, Queue (linear data structure), Trees are a non-linear data structure. To be precise, it's a hierarchical data structure.



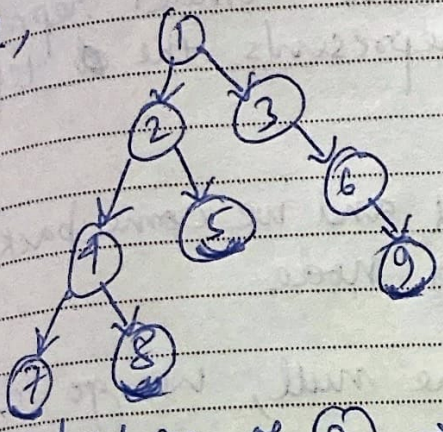
Ancestors :- The nodes on a higher level from that node.
E.g. ancestors of (7) → (4, 2, 1).
ancestors of (6) → (3, 1).

Level :- There are levels in a tree data structure. The top most is the root (level 1) and then level 2, and so on.

Depth :- With level, there is an associated term called depth. Most of the times, there are in sync, if we start from level 1.
So, for e.g. (4), (5), (6) falls in level 3 and their depth is also 3.

Subtree :- There are left and right sub-trees to a node.
For e.g. left subtree of (2) is → (4) → (7)
right sub-tree of (2) is → (5) → (8)

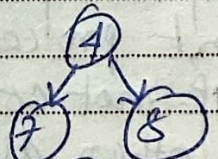
Eg,



leaf node → nodes with no child.

(a) children of (2) → (4), (5), [redacted] (immediate child)
(b) No. of leaves → 4 (7, 8, 5, 9)
(c) Parent of (3) → (1)
(d) Level of (5) → 3.
(e) Subtrees of 2 & 6

For (2), left subtree →



(4, 7, 8)

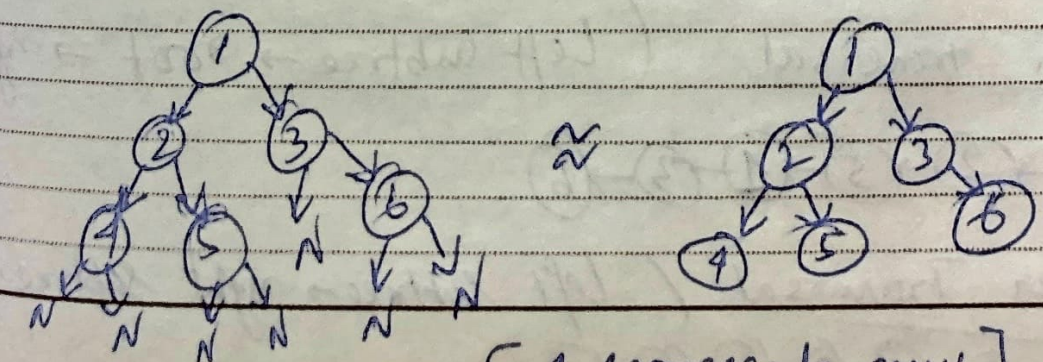
right subtree → (5).

For (6), left subtree → null
right subtree → (9)

(f) Ancestors of (4) → (2, 1)

Build Tree Pre-order

1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1



[-1 represents null].

03 August
32nd Wk • 215-150
Wednesday

2022
AUGUST
1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31
M T W T F S S M T W T F S S

So, we place the root first, then the number represents it's left child. The number next represents the left child to the left child.

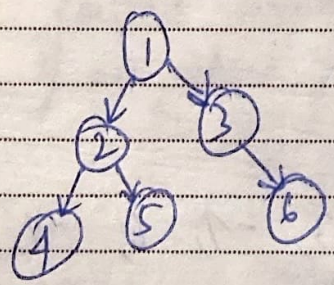
when we face -1, it means null and we come back to one level up and go for the other node.

when both child nodes are null, we go 2 levels up from the null.

when we reach a level, we see whether both the left and right sub-trees are completed or not.
If completed, we move one level up and then start proceeding with the next number.
If not completed, we go to the sub-tree in the other direction.

we always start with the left-subtree and only when that's completed, we move to the right subtree.

PreOrder Traversal (Root, left Sub-tree, right Subtree)



1, 2, 4, 5, 3, 6.

In order traversal (Left Subtree → root → right Subtree)

4 → 2 → 5 → 1 → 3 → 6.

Post-order Traversal (left subtree → right subtree → root)

4 → 5 → 2 → 6 → 3 → 1.

Level order Traversal

Here, we shall go level wise. So,

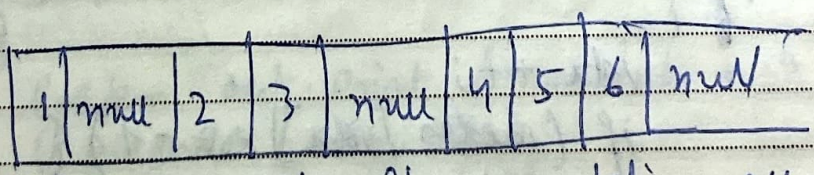
1
2 3 → data shall be printed in this manner.
4 5 6

✓ In the last 3 traversal methods, namely Pre-order, In-order and Post-order Traversal, we ~~use~~ shall utilize recursion. Also, we can say that we are using the concept of DFS or ~~Breadth First search~~ Depth First Search as we are going down to the leaf nodes.

✓ In level order Traversal, we shall do the following —

(i) we shall use the concept of Queue for FIFO algorithm.
(ii) we shall use the concept of BFS or Breadth First Search.

Process :-



✓ we shall enter a null after completing all nodes of that level.

✓ Now, when we pop out elements,

1 null
2 3 null → As soon as we face null, it means we shall go to the next and print the subsequent levels.
4 5 6 null

✓ Time complexity = $O(n)$ → as we are traversing every node of the tree.

05 Friday
August
32nd Wk • 217-148

2022
AUGUST
1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31
M T W T F S S M T W T F S S

Build Tree from given Pre-order Sequences

```
Public class BinaryTree4T {

    Static class Node {
        int data;
        Node left;
        Node right;

        Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }

    Static class BinaryTree {
        Static int idx = -1;
        Public Static Node buildTree(int nodes[]) {
            idx++;
            if (nodes[idx] == -1) {
                return null;
            }

            Node newNode = new Node(nodes[idx]);
            newNode.left = buildTree(nodes);
            newNode.right = buildTree(nodes);
            return newNode;
        }
    }
```

```
Public static void main (String args[])
{
    int nodes[] = {1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1};

    BinaryTree tree = new BinaryTree();
    Node root = tree.buildTree(nodes);
    System.out.println(root.data);
}
}
```

** Tree Traversals **

(a) Pre-order

```
Public static void preOrder (Node root) {
    if (root == null) {
        System.out.print(-1 + " ");
        return;
    }
    System.out.print(root.data + " ");
    preOrder(root.left);
    preorder(root.right);
}
```

(b) In-order

```
Public static void inorder (Node root) {
    if (root == null) {
        System.out.print(-1 + " ");
        return;
    }
    inorder(root.left);
    System.out.print(root.data + " ");
    inorder(root.right); }
```

(c) Postorder

```
8   Public static void postorder(Node root) {
9       if (root == null) {
            System.out.print(-1 + " ");
10          return;
        }
11      Postorder(root.left);
        postorder(root.right);
12      ~~postorder(~~ System.out.print(root.data + " ");
    }
```

1(d) level order

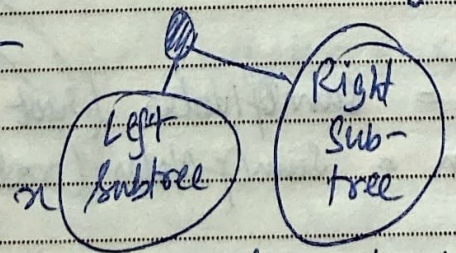
```
    Public static void levelorder(Node root) {
2       if (root == null) {
3           return;
        }
4       Queue<Node> q = new LinkedList<>();
        q.add(root);
5       q.add(null);
6       while (!q.isEmpty()) {
            Node curr = q.remove();
            if (curr == null) {
                System.out.println();
                if (q.isEmpty()) {
                    break;
                } else {
                    q.add(null);
                }
            }
        }
```

```
else {
    System.out.print(curr.data + " ");
    if (curr.left != null) {
        q.add(curr.left);
    }
    if (curr.right != null) {
        q.add(curr.right);
    }
}
}
```

3) Count of Nodes

We shall solve this kind of problem, especially when we need to count or to track any property for every node, we can use recursion. Many problems of Trees can be solved using the concept of recursion.

Algorithm –



So, total = x + y + 1.

✓ Count total no. of nodes in the left sub-tree (x)
✓ Count total no. of nodes in the right sub-tree. (y)
✓ Total = x + y + 1 [∵ 1 for the root node].

✓ Now do the same process recursively for every node in the left sub-tree and right sub-tree.

✓ Time complexity = O(n). The code touches every node once, hence O(n) is the time taken.

09 August
33rd Wk • 221-144
Tuesday

2022 AUGUST
1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31
M T W T F S S M T W T F S S

→ Public static int countOfNodes (Node root) {

8 if (root == null) {
return 0;
9 }
int leftNodes = countOfNodes (root.left);
10 int rightNodes = countOfNodes (root.right);
return (leftNodes + rightNodes + 1);
11 }

12 ** 2) Sum of nodes **

Time complexity = O(n). Here also, the code touches every
1 node once to get the value.

2 → Public static int sumOfNodes (Node root)
{
3 if (root == null) {
return 0;
4 }

5 int leftSum = sumOfNodes (root.left);
int rightSum = sumOfNodes (root.right);
6
return (leftSum + rightsum + root.data);
}

** 3) Height of a Tree **

✓ Here, also we shall recursion.
✓ Now, calculate the height of the left subtree & right subtree.
✓ (Maximum height b/w those sub-trees) + 1 (for root)
= total height of the tree.

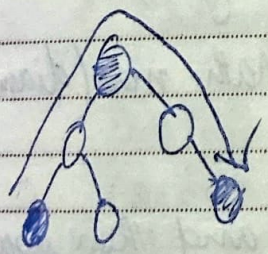
→ Public static int height (Node root) {

```
    if (root == null) {
        return 0;
    }
    int leftHeight = height (root.left);
    int rightHeight = height (root.right);
    return Math.max(leftHeight, rightHeight) + 1;
}
```

8) Diameter of a Tree

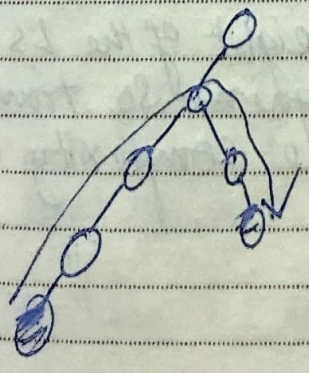
→ It is the number of nodes in the longest path between any 2 nodes. Now, to understand better, we shall consider 2 cases —

Case 1 : Diameter through Root



diameter = 5.

Case 2 :

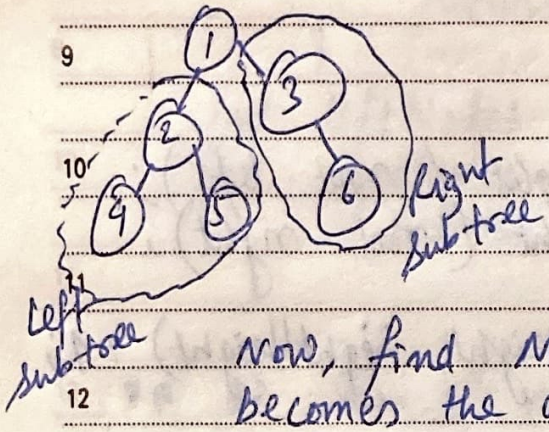


diameter = 6.

na node, d = 5 or d = 4, etc.
So, we choose farthest nodes, so that the path becomes the longest, and it doesn't include the root

Now, to solve the cases, we shall have 2 approaches :—

Approach 1 : O(N^2)



Diam 1 = find largest diameter in left subtree.
Diam 2 = find largest diameter in right subtree
Diam 3 = (Left Sub-tree depth) + (right sub-tree depth) + 1.

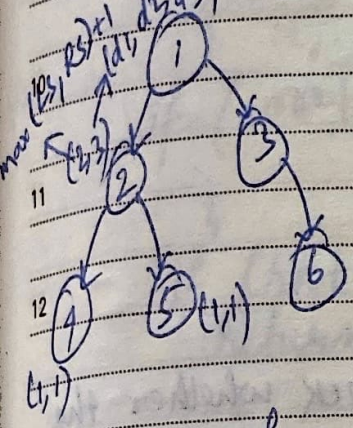
now, find Max (Diam 1, Diam 2, Diam 3). That becomes the actual diameter of our tree.

```
public static int diameter (Node root) {
    if (root == null) {
        return 0;
    }
    int diam1 = height (root.left) + height (root.right) + 1;
    int diam2 = diameter (root.left);
    int diam3 = diameter (root.right);

    return Math.max (diam1, Math.max (diam2, diam3));
}
```

Note :— we touched every node once and then again when we are calculating the height of the LS and RS, we are traversing it again. So, traversing 'n' nodes 'n' times, So, time complexity = n^2.

Approach 2 : O(n)

To optimize, we shall calculate the height parallely in addition to calculating the diameter. So, we don't need to call the function again and traverse 'n' times. That's how, we shall optimize the time complexity.



So, we shall calculate 2 things for every node -

✓ height, diameter.

For (2) → (2,3) → (diam1, diam2, diam3)

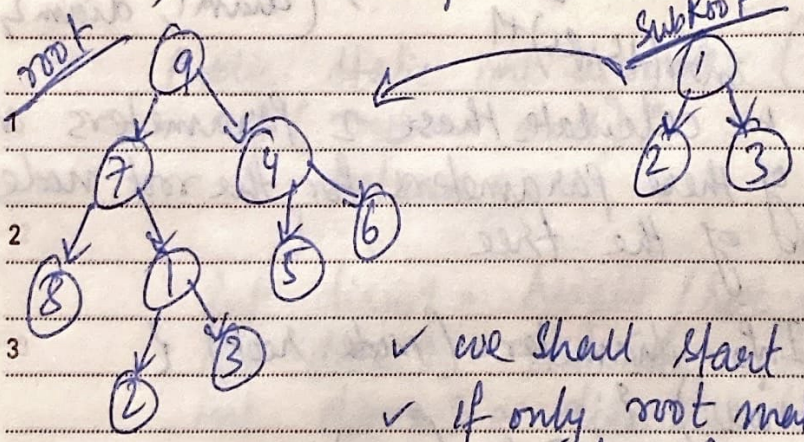
max(LS, RS)+1

Similarly, for all nodes we calculate these 2 parameters and finally, the Max of these parameters for the root node shall be our diameter of the tree.

```
Public static TreeInfo diameter (Node root) {
    if (root == null) {
        return new TreeInfo (0, 0);
    }
    TreeInfo leftTI = diameter (root.left);
    TreeInfo rightTI = diameter (root.right);
int myHeight = Math.max (leftTI.height, rightTI.height)+1;

    int diam1 = leftTI.height + rightTI.height + 1;
    int diam2 = leftTI.diam;
    int diam3 = rightTI.diam;

int myDiam = Math.max (diam1, Math.max (diam2, diam3));

    return new TreeInfo (myHeight, myDiam);
}
```

```
static class TreeInfo {
    int ht;
    int diam;

    TreeInfo (int ht, int diam) {
        this.ht = ht;
        this.diam = diam;
    }
}
```

** 8) Subtree of another tree **



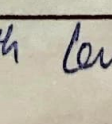
(check whether the given tree is a sub-tree of another bigger tree or not)

✓ we shall start by matching the root.
✓ if only root matches, we start matching the left and the right subtree.

```
// defining a binary tree node.
Public class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode () { }
    TreeNode (int val) { this.val = val; }
    TreeNode (int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
```

```
class Solution
{
Public boolean isIdentical (TreeNode root, TreeNode
                            SubRoot) {
if (SubRoot == null && root == null) {
    return true;
}
if (root == null || SubRoot == null) {
    return false;
}
if (root.val == SubRoot.val) {
return isIdentical (root.left, SubRoot.left) &&
    isIdentical (root.right, SubRoot.right);
}
return false;
}

Public boolean isSubtree (TreeNode root,
                          TreeNode SubRoot) {
if (SubRoot == null) {
    return true;
}
if (root == null) {
    return false;
}
if (isIdentical (root, SubRoot)) {
    return true;
}
return isSubtree (root.left, SubRoot) ||
    isSubtree (root.right, SubRoot);
}
```

Homework :- Find the Sum at Kth level.



If K=2, Sum = 5.