PDP ONLINE

DSA – Data Structures

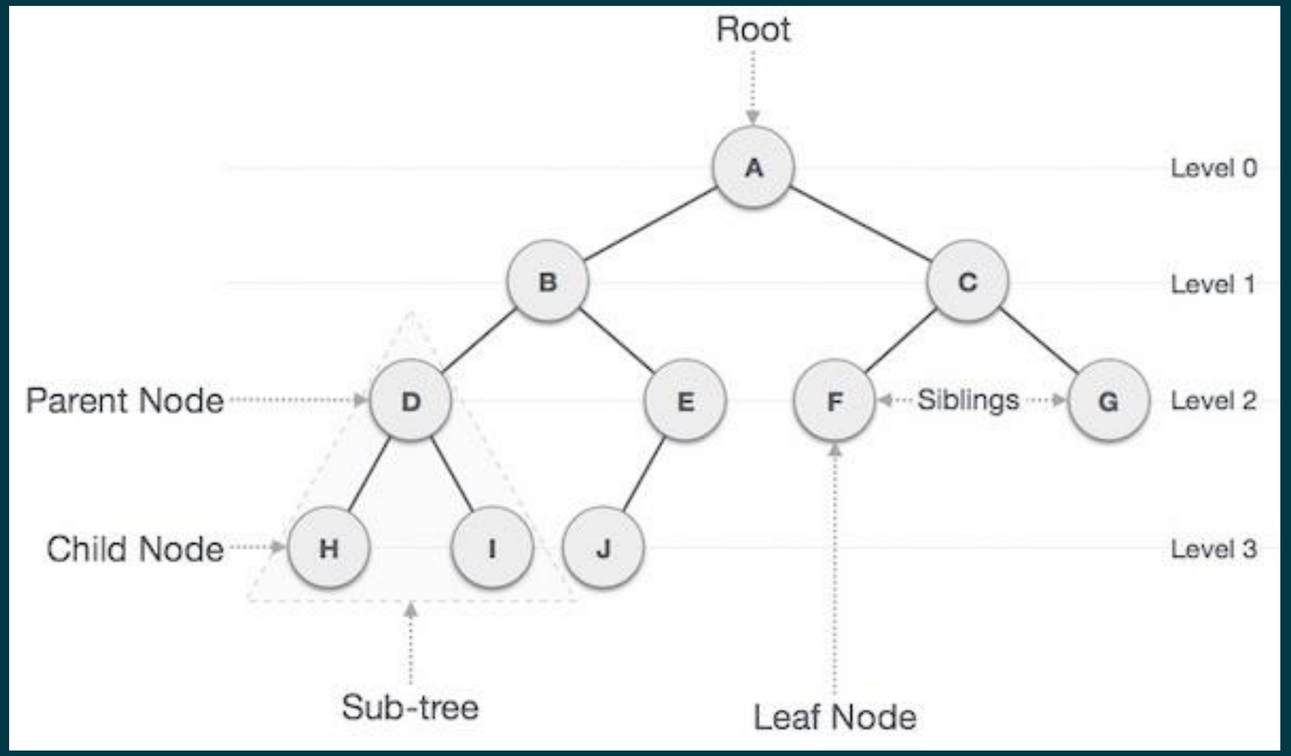
Tree

Course Planning

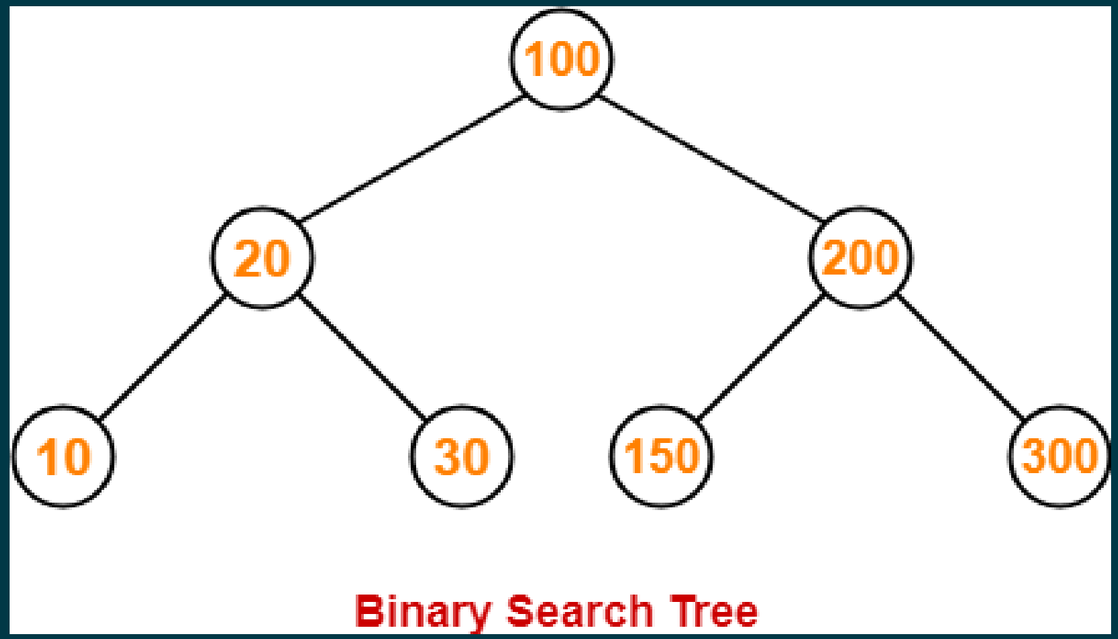
Algorithms	Data Structures	Algorithmic Approaches	Interview Practices
1.Introduction	1.Asymptotic Analysis	1.Search Algorithms	1.In-place Reversal
2.Number 1	2.Dynamic Array	2.Sort Algorithms	2.Two Heaps
3.Number 2	3.LinkedList	3.Dac Algorithms	3.Subsets
4.String 1	4.Stack	4.Recursion	4.Modified BS
5.String 2	5.Queue	5.Sliding Window	5.Bitwise XOR
6.Array 1	6.HashTable	6.Two Pointers	6.Top 'K' Elements
7.Array 2	7.Tree	7.Fast & Slow	7.K-way Merge
8.Matrix	8.Trie	8.Cyclic Sort	8.Knapsack Problem
9.DP 1	9.Directed Graph	9.Breadth First Search	9.Topological Sort
10.DP 2	10.Undirected Graph	10.Depth First Search	10.Mock Interview

https://kurbanovxurshid.medium.com/data-structures-and-algorithms-course-plan-1ab912ec1e17

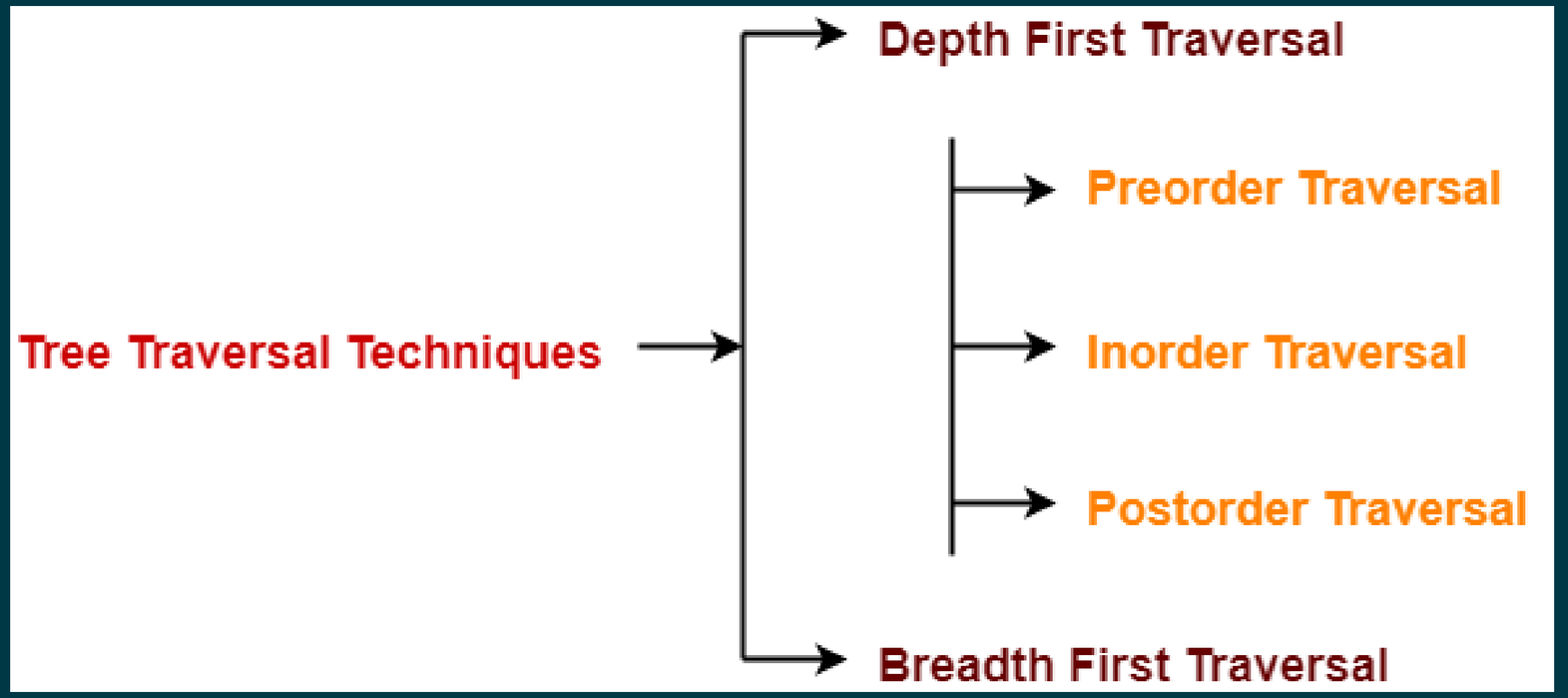
Binary Tree


Root
A
B
C
D
E
F
G
H
I
J
Level 0
Level 1
Level 2
Level 3
Parent Node
Siblings
Child Node
Sub-tree
Leaf Node

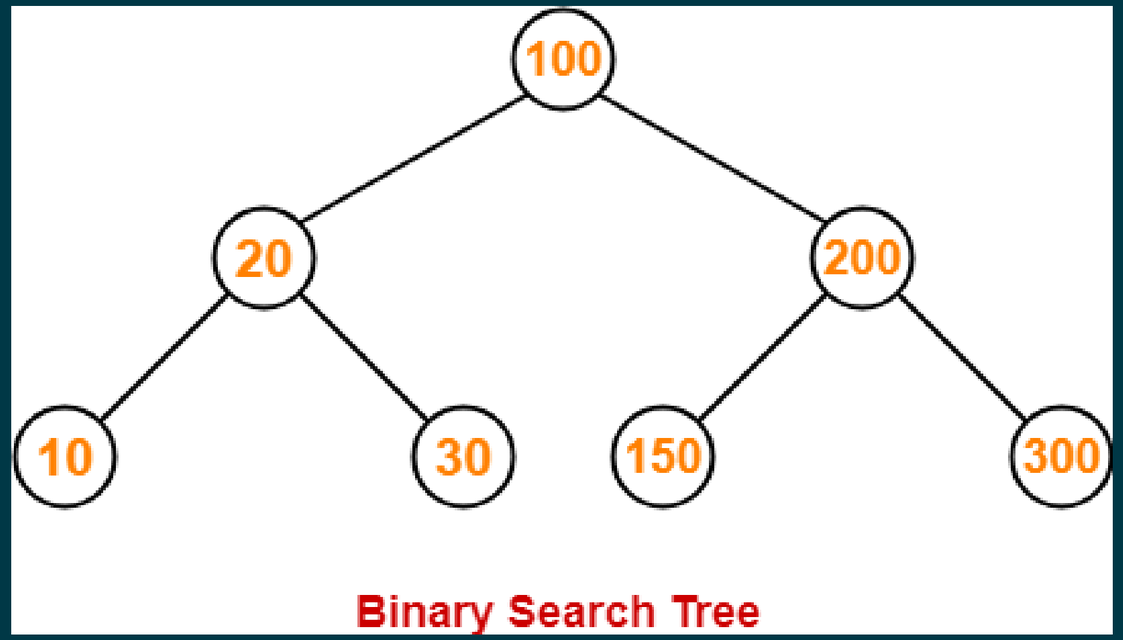
Binary Search Tree



Tree Traversal



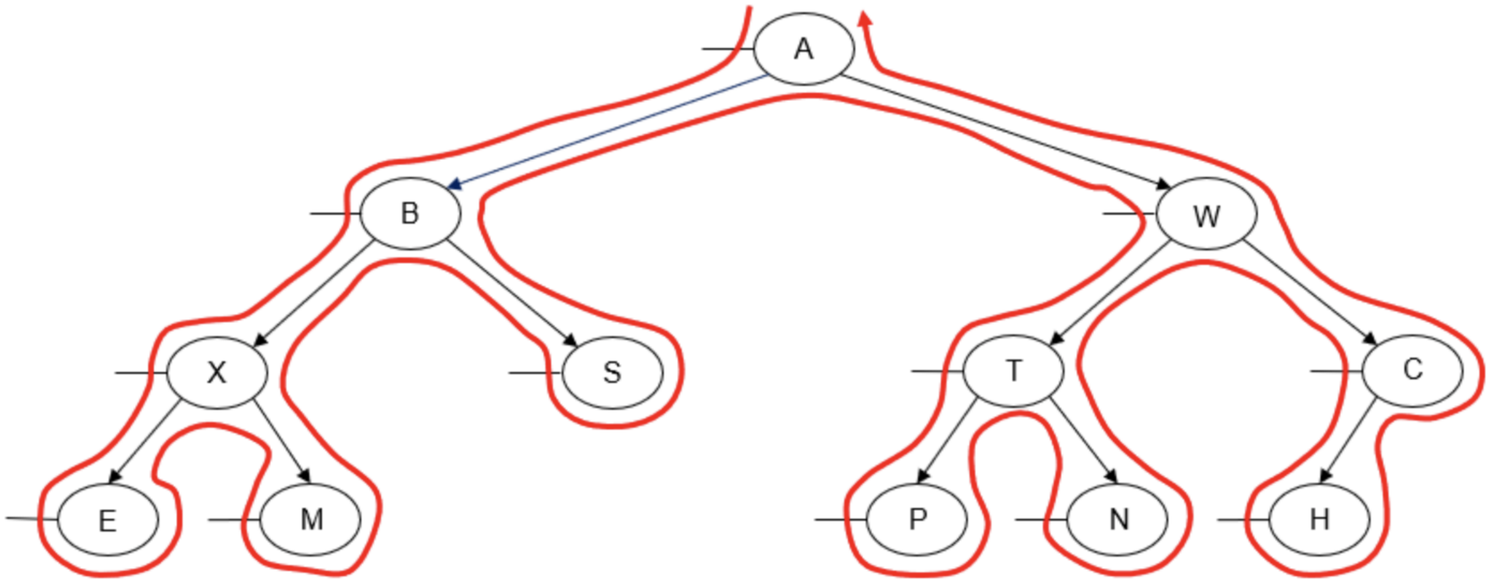
Depth & Breadth



Depth First Traversal
Pre-Order: 100 , 20 , 10 , 30 , 200 , 150 , 300 (Root-Left-Right)
In-Order: 10 , 20 , 30 , 100 , 150 , 200 , 300. (Left-Root-Right)
Post-Order: 10 , 30 , 20 , 150 , 300 , 200 , 100 (Left-Right-Root)

Breadth First Traversal
Level-Order: 100 , 20 , 200 , 10 , 30 , 150 , 300

Pre-Order Traversal

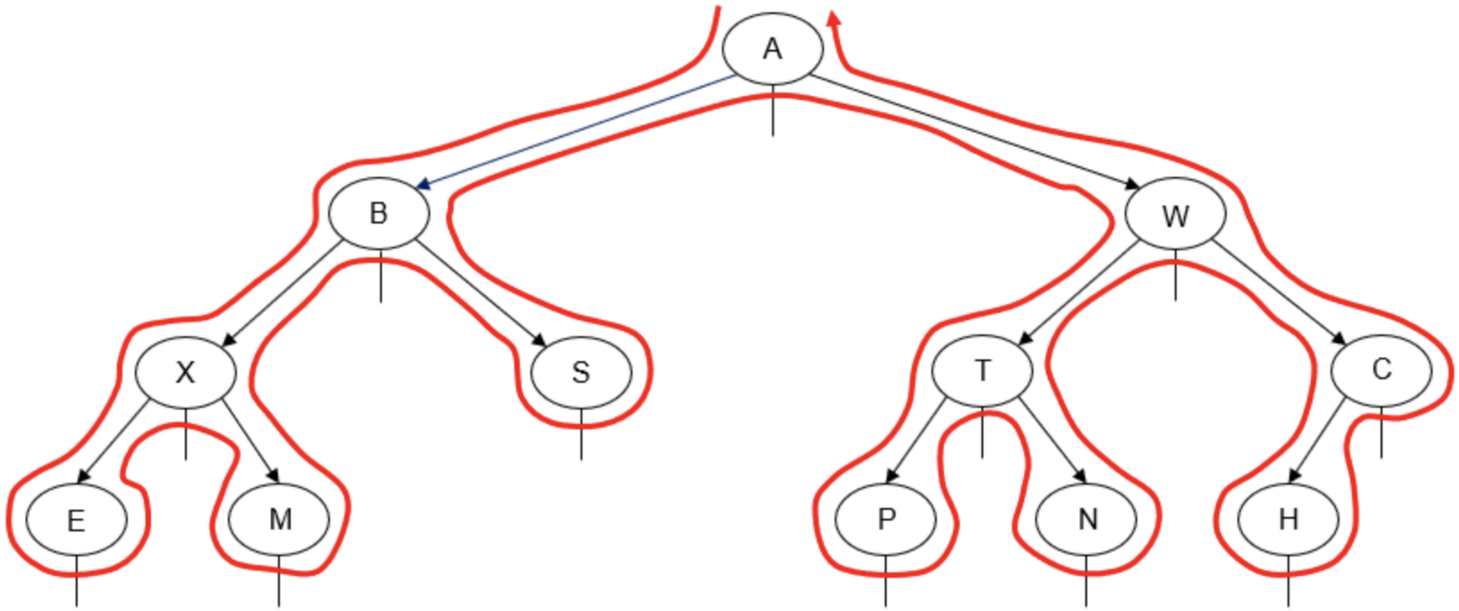


Printing the value of each node as we "visit" it, we get the following output:

```
A B X E M S W T P N C H
```

In-Order Traversal

Here's an example of a left-to-right inorder traversal of a binary tree:

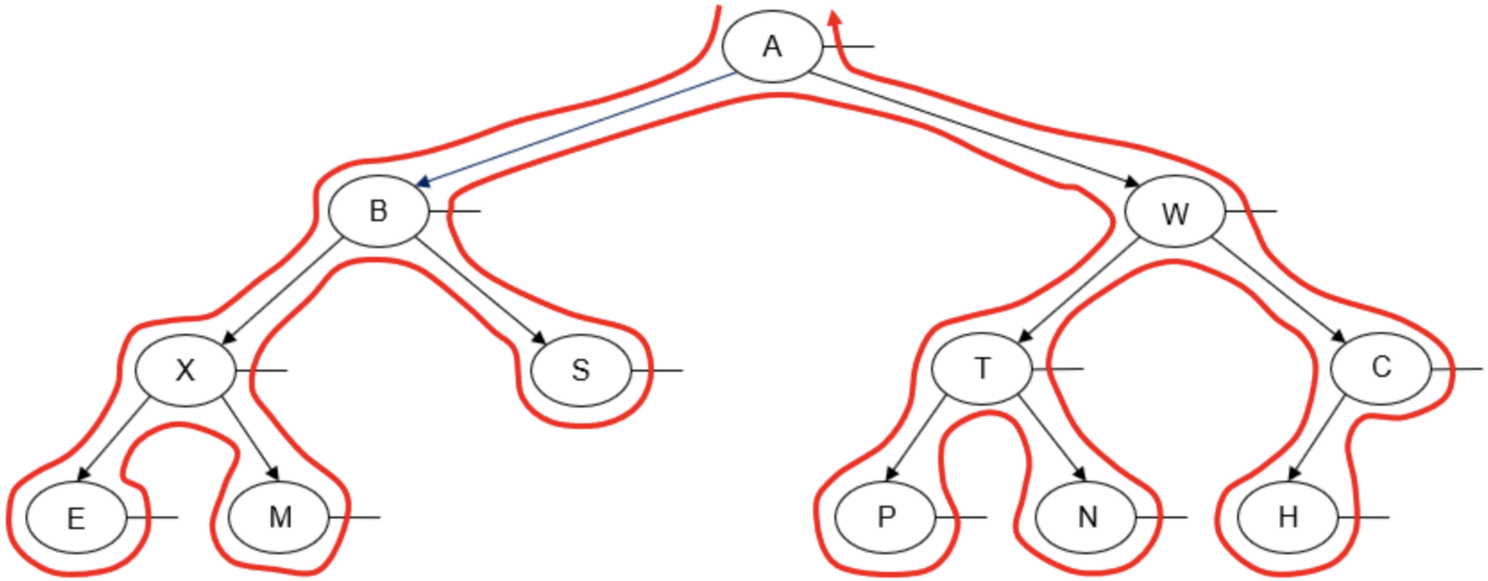


Printing the value of each node as we "visit" it, we get the following output:

```
E X M B S A P T N W H C
```

Post-Order Traversal

Here's an example of a left-to-right postorder traversal of a binary tree:

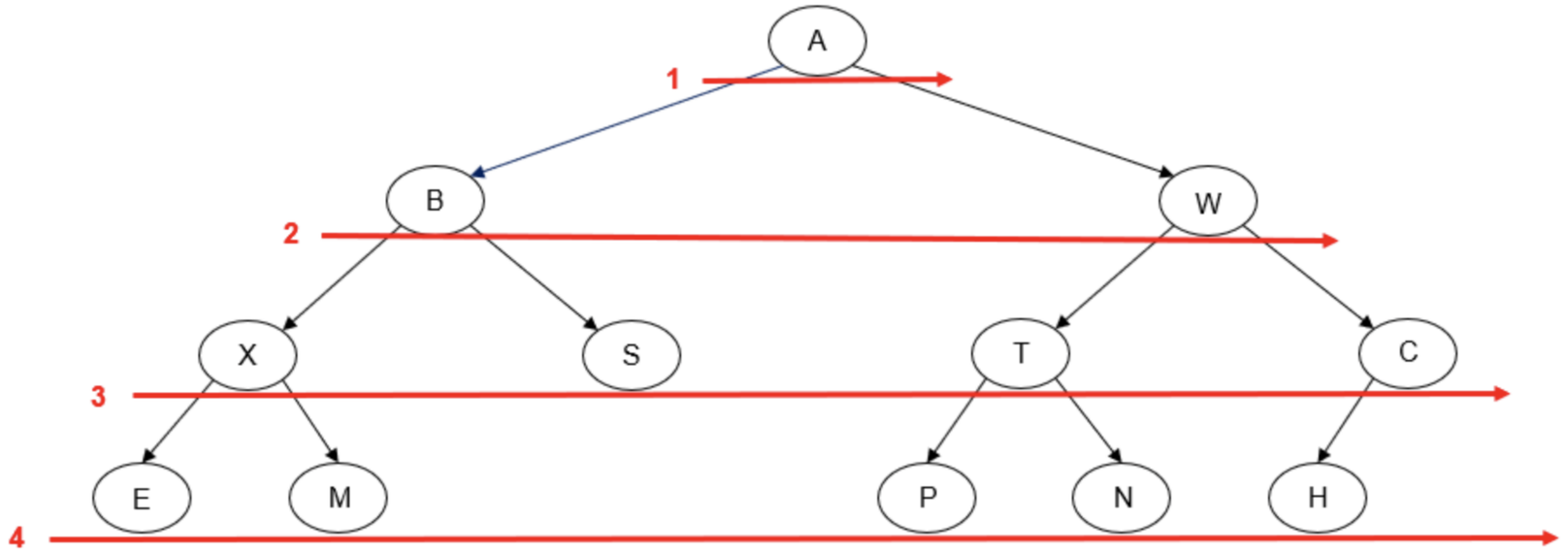


Printing the value of each node as we "visit" it, we get the following output:

```
E M X S B P N T H C W A
```

Level-Order Traversal

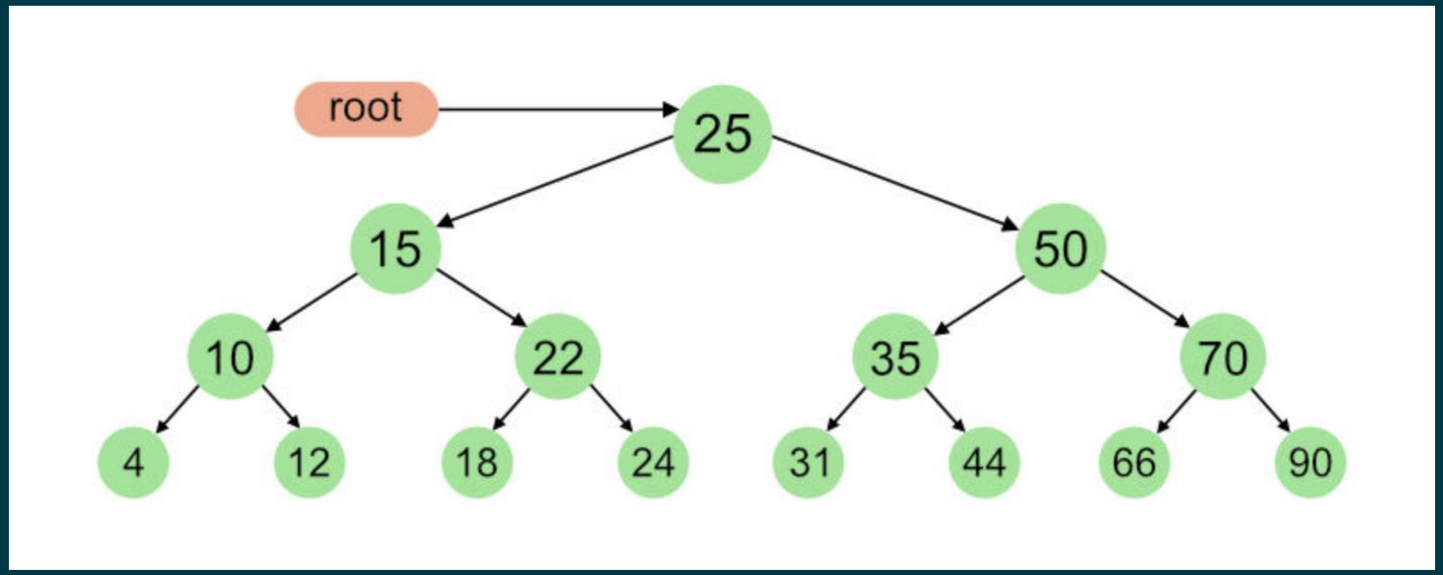
Here's an example of a left-to-right level order traversal of a binary tree:



Printing the value of each node as we "visit" it, we get the following output:

```
A B W X S T C E M P N H
```

Find Traversals



BST

```java
public class BinaryTree {

    private class Node{
        private int value;

        private Node left;
        private Node right;

        public Node(int value) {
            this.value = value;
        }
    }

    private Node root;
```

insert

```
//Time Complexity : O(log N)
public void insert(int item) {
    if(root ==  null) {
        root = new Node(item);
        return;
    }

    var current = root;
    while(true) {
        if(item > current.value) {// right
            if(current.right ==  null) {
                current.right = new Node(item);
                break;
            }
            current = current.right;
        }else {//left
            if(current.left ==  null) {
                current.left = new Node(item);
                break;
            }
            current = current.left;
        }
    }

}
```

traversalPreOrder

```java
//Time Complexity : O(N)
public void traversePreOrder() {
    traversePreOrder(root);
}

private void traversePreOrder(Node root) {
    if(root == null) return;

    System.out.println(root.value);
    traversePreOrder(root.left);
    traversePreOrder(root.right);
}
```

traversalInOrder

```java
//Time Complexity : O(N)
public void traverseInOrder() {
    traverseInOrder(root);
}

private void traverseInOrder(Node root) {
    if(root == null) return;

    traverseInOrder(root.left);
    System.out.println(root.value);
    traverseInOrder(root.right);
}
```

traversalPostOrder

```java
//Time Complexity : O(N)
public void traversePostOrder() {
    traversePostOrder(root);
}

private void traversePostOrder(Node root) {
    if(root == null) return;

    traversePostOrder(root.left);
    traversePostOrder(root.right);
    System.out.println(root.value);
}
```

height

```java
//Time Complexity: O(n)
public int height() {
    return height(root);
}

private int height(Node root) {
    if(root == null)
        return 0;

    int lheight = height(root.left);
    int rheight = height(root.right);

    if(lheight > rheight)
        return (lheight + 1);
        else
    return rheight + 1;
}
```

traversalLevelOrder

```java
//Time Complexity : O(N)
public void traverseLevelOrder() {
    int h = height();

    for(int i=1; i<=h; i++) {
        printLevelOrder(root, i);
    }
}

private void printLevelOrder(Node root, int level) {
    if(root == null) return;
    if(level == 1) {
        System.out.println(root.value);
    }else {
        printLevelOrder(root.left, level-1);
        printLevelOrder(root.right, level-1);
    }
}
```

find

```
//Time Complexity : O(log N)
public boolean find(int item) {
    var current = root;
    while(current != null) {
        if(item < current.value) {
            current = current.left;
        }else if(item > current.value) {
            current = current.right;
        }else {
            return true;
        }
    }

    return false;
}
```

Task 1
Darsda o`tilgan BinaryTree da hamma barglarini chop qiladigan imkoniyatni qo`shing.

public void printAllLeaves()

Task 2
Darsda o`tilgan BinaryTree da gi eng kichik elementni topadigan imkoniyatni qo`shing.

public void min()

Task 3
Darsda o`tilgan BinaryTree da gi eng katta elementni topadigan imkoniyatni qo`shing.

public void max()

Task 4
Darsda o`tilgan BinaryTree da berilgan Tree ni Binary Search Tree ekanligini yoki emasligini aniqlaydigan imkoniyat qo`shing.

public void isBinarySearchTree()

Task 5
Darsda o`tilgan BinaryTree da K-inchi levelda turgan Node larni chop qiladigan dastur yozing.

public void printNodesAtDistance(int k)