DSA Binary Search Tree

Prof-: Gautam Singh

TREES – Non-Linear Data Structure

What is a Tree?

A **Tree** is a **non-linear data structure** that stores data in a **hierarchical** manner.
It consists of **nodes** connected by **edges** and has a **root node**.

Tree Terminology

Term	Description
Node	Each element in a tree
Root	Topmost node of the tree
Parent	Node having child nodes
Child	Node derived from parent
Leaf	Node with no children
Edge	Connection between nodes
Height	Longest path from root to leaf
Degree	Number of children of a node

Types of Trees

- General Tree
- A node can have **any number of children**
- No restriction on structure
- **Example:** File System

- Binary Tree
- Each node has **at most 2 children**
- Children are called:
  - Left Child
  - Right Child

- Binary Search Tree (BST)
  - A **special binary tree** where:
  - `Left child < Root < Right child`

Binary Tree Node

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

Inorder Traversal (L → Root → R)

Tree Traversals

Tree traversal means **visiting each node exactly once**.

```
void inorder(Node root) {
if (root != null) {
inorder(root.left);
System.out.print(root.data + " ");
inorder(root.right);
}
}
```

Preorder Traversal (Root → L → R)

```
void preorder(Node root) {
if (root != null) {
System.out.print(root.data + " ");
preorder(root.left);
preorder(root.right);
}
}
```

Postorder Traversal (L → R → Root)

```
void postorder(Node root) {
if (root != null) {
postorder(root.left);
postorder(root.right);
System.out.print(root.data + " ");
}
}
```

Insertion in BST

```
Node insert(Node root, int key) {
   if (root == null) {
      return new Node(key);
   }

   if (key < root.data)
      root.left = insert(root.left, key);
   else if (key > root.data)
      root.right = insert(root.right, key);

   return root;
}
```

Searching in BST

```
boolean search(Node root, int key) {
   if (root == null)
      return false;

   if (root.data == key)
      return true;

   if (key < root.data)
      return search(root.left, key);
   else
      return search(root.right, key);
}
```

Deletion in BST

```
Node delete(Node root, int key) {
   if (root == null)
      return root;
   if (key < root.data)
      root.left = delete(root.left, key);
   else if (key > root.data)
      root.right = delete(root.right, key);
   else {       // one child or no child
      if (root.left == null)
         return root.right;
      else if (root.right == null)
         return root.left;      // two children
      root.data = minValue(root.right);
      root.right = delete(root.right, root.data);    }
   return root;   }
```

```
int minValue(Node root) {
int min = root.data;
while (root.left != null) {
min = root.left.data;
root = root.left;
}
return min;
}
```

```
class BST {
public static void main(String[] args) {
BST tree = new BST();
tree.root = tree.insert(tree.root, 50);
tree.insert(tree.root, 30);
tree.insert(tree.root, 70);
tree.insert(tree.root, 20);
tree.insert(tree.root, 40);
System.out.print("Inorder: ");
tree.inorder(tree.root);
System.out.println("\nSearch 40: " + tree.search(tree.root, 40));
tree.root = tree.delete(tree.root, 20);
System.out.print("After Deletion: ");
tree.inorder(tree.root);
}
}
```

Binary Search

```
int binaySearch(int arr[] , int x){
    int left = 0;
    int right = arr.length-1;
    while(left<=right){
      int mid = left + (right-left)/2;
      if(arr[mid]== x){
        return mid;
      }
      else if( arr[mid]<x){
        left = mid+1;
      }
      else{
        right= mid -1;
      }
    }
    return left;
  }
```