



DSA SERIES

- Learn Coding



Topic to be Covered today

Binary Search Tree



LETS START TODAY'S LECTURE



A **Binary Search Tree (BST)** is a binary tree with an additional property:

👉 For every node:

- All nodes in its **left subtree** have values **smaller** than the node's value.
- All nodes in its **right subtree** have values **greater** than the node's value.
- Both left and right subtrees must also be BSTs.

Structure of a Node

Each node has:

- A value (data)
- A pointer to the left child
- A pointer to the right child



```
struct Node {  
    int data;  
    Node* left;  
    Node* right;  
    Node(int val) {  
        data = val;  
        left = right = NULL;  
    }  
};
```



Why Use BST?

BST helps in **fast searching, insertion, and deletion**.

- Average case: **$O(\log n)$** (like binary search)
- Worst case: **$O(n)$** (if tree becomes skewed)

Use cases:

- Implementing sets & maps (C++ STL `set` and `map` use balanced BST internally)
- Searching in large datasets
- Autocomplete features
- Database indexing



Operations in BST

(a) Insertion

Insert values while maintaining BST rules.

Code :

```
Node* insert(Node* root, int val) {
    if (root == NULL) return new Node(val);
    if (val < root->data)
        root->left = insert(root->left,
val);
    else
        root->right = insert(root->right,
val);
    return root;
}
```



Searching

Search like binary search:

- If value < root → search left
- If value > root → search right
- If value == root → found

```
bool 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);  
    return search(root->right, key);  
}
```




Traversals

- **Inorder (LNR)** → gives **sorted order** of elements in BST
- **Preorder (NLR)** → useful for copying tree
- **Postorder (LRN)** → useful for deleting tree

Example (Inorder):

```
void inorder(Node* root) {  
    if (root == NULL) return;  
    inorder(root->left);  
    cout << root->data << " ";  
    inorder(root->right);  
}
```

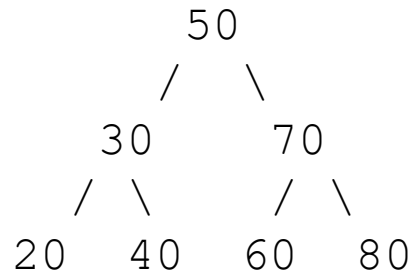
Deletion in BST

When deleting a node from a BST, we must make sure the **BST property** is preserved.

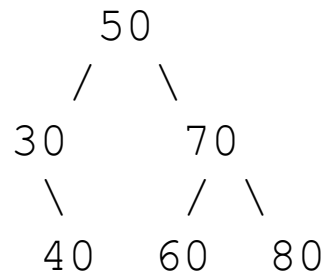
✓ 3 Cases of Deletion

1. Case 1: Node is a Leaf Node (no children)

- Simply delete the node.
- Example: Deleting 20 from:

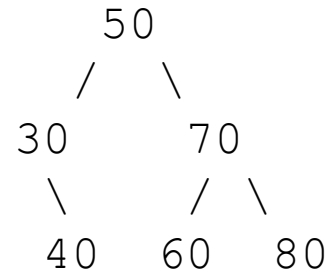


Remove 20, tree becomes:

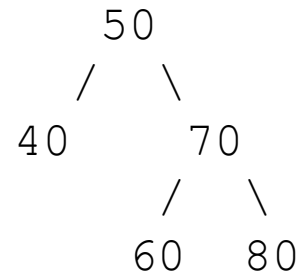


2. Case 2: Node has One Child

- Replace the node with its child.
- Example: Deleting 30 (which has only one child \rightarrow 40):



After deleting 30:



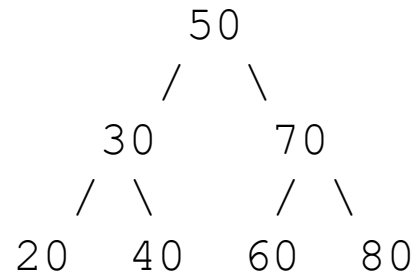
3. Case 3: Node has Two Children

👉 This is the trickiest case.

Steps:

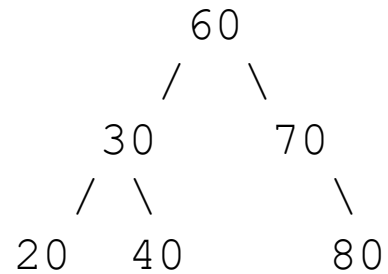
1. Find **inorder successor** (minimum node in the right subtree)
OR **inorder predecessor** (maximum in the left subtree).
2. Replace the node's value with the successor/predecessor value.
3. Delete that successor/predecessor node.

- ◆ Example: Delete 50 (root, has two children):



- Inorder successor of 50 = **60** (smallest in right subtree).
- Replace 50 with 60.
- Then delete 60 from the right subtree.

Result:



C++ Code for Deletion:

```
Node* deleteNode(Node* root, int key) {
    if (root == NULL) return root;

    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    }
    else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    }
    else {
        // Node found
        // Case 1 & 2: Node with 0 or 1 child
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
```

```
delete root;
    return temp;
}

// Case 3: Node with 2 children
// Get inorder successor (min in right subtree)
Node* temp = root->right;
while (temp->left != NULL) {
    temp = temp->left;
}
// Copy inorder successor's value
root->data = temp->data;

// Delete inorder successor
root->right = deleteNode(root->right, temp->data);
}
return root;
}
```

108. Convert Sorted Array to Binary Search Tree

```
class Solution {
public:
    TreeNode* helper(vector<int>& nums, int start, int end) {
        if (start > end)
            return NULL;

        int mid = start + (end - start) / 2;

        TreeNode* root = new TreeNode(nums[mid]);

        root->left = helper(nums, start, mid - 1);
        root->right = helper(nums, mid + 1, end);

        return root;
    }
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return helper(nums, 0, nums.size() - 1);
    }
};
```


501. Find Mode in Binary Search Tree

```
class Solution {
public:
    // Helper function for inorder traversal
    and counting frequencies
    void helper(TreeNode* root, vector<int>&
ans, int& currentVal,
                int& currentCount, int&
maxCount) {
        if (root == nullptr)
            return; // If the node is null,
return.

        // Traverse the left subtree
        helper(root->left, ans, currentVal,
currentCount, maxCount);
```

```
// Process the current node
    if (root->val == currentVal) {
        // If the current value is the same as the previous, increase the
        // count
        currentCount++;
    } else {
        // If the value changes, reset the count for the new value
        currentVal = root->val;
        currentCount = 1;
    }

// Update the answer and maximum count
if (currentCount > maxCount) {
    // If we find a new maximum count, clear the answer and add the new
    // mode
    maxCount = currentCount;
    ans.clear();
}
```

```

ans.push_back(root->val);
    } else if (currentCount == maxCount) {
        // If the current count matches the maximum, add it to the answer
        ans.push_back(root->val);
    }

    // Traverse the right subtree
    helper(root->right, ans, currentVal, currentCount, maxCount);
}

```

```

vector<int> findMode(TreeNode* root) {
    vector<int> ans; // To store the modes
    int currentVal =
        INT_MIN; // To keep track of the current value during traversal
    int currentCount = 0; // Count of occurrences of the current value
    int maxCount = 0; // Maximum count of any value
}

```

```
        // Call the helper function for in-order traversal
        helper(root, ans, currentVal, currentCount, maxCount);

        return ans; // Return the modes
    }
};
```



Learn coding

THANK YOU