LEETCODE 669

📘 LeetCode 669 – Trim a Binary Search Tree

🧠 Problem Summary:

Given a **Binary Search Tree (BST)** and a range `[low, high]`, trim the tree such that all its elements lie within this range.

- The **relative structure of nodes that remain** must be unchanged.
- Return the **new root** of the trimmed tree.

✅ Constraints:

- 1 ≤ Number of nodes ≤ 10^4
- 0 ≤ Node.val ≤ 10^4
- Each node has **unique** values
- Tree is a valid BST
- `0 ≤ low ≤ high ≤ 10⁴`

👨‍💻 Intuition:

Since this is a **Binary Search Tree**, we know:

- Left subtree < root
- Right subtree > root

So we can use this property to:

- Recursively discard nodes not in range `[low, high]`
- Adjust pointers accordingly

🛠️ Approach:

We:

1. Create a dummy node pointing to the root
2. Trim all nodes **less than low** from the left subtree
3. Trim all nodes **greater than high** from the right subtree
4. Recursively repeat for left and right children

🧾 Code Explanation (If-Else Based):

```
class Solution {
public:
    // Helper function to trim the BST recursively
    void trim(TreeNode* root, int lo, int hi) {
        if (root == NULL) return;

        // Prune left child
        while (root→left != NULL) {
            if (root→left→val < lo) {
                // Discard left child and move to its right
                root→left = root→left→right;
            } else if (root→left→val > hi) {
                // Discard left child and move to its left
                root→left = root→left→left;
            } else {
                // Value is in range, stop trimming left
                break;
            }
        }

        // Prune right child
        while (root→right != NULL) {
            if (root→right→val > hi) {
                // Discard right child and move to its left
                root→right = root→right→left;
            } else if (root->right->val < lo) {
                // Discard right child and move to its right
                root->right = root->right->right;
            } else {
                // Value is in range, stop trimming right
                break;
            }
        }

        // Recursively trim remaining subtrees
        trim(root->left, lo, hi);
        trim(root->right, lo, hi);
    }

    // Main function
    TreeNode* trimBST(TreeNode* root, int lo, int hi) {
        // Dummy node to simplify root adjustments
        TreeNode* dummy = new TreeNode(INT_MAX);
        dummy->left = root;

        // Start trimming from dummy node
        trim(dummy, lo, hi);

        // Return the updated root
        return dummy->left;
    }
};
```

🔍 Key Observations:

- `while` loops are used instead of recursion to avoid going too deep unnecessarily
- Dummy node ensures root-level pruning without special cases
- The final root may change, so dummy helps retain pointer easily

🌳 Example:

Input:

```
Tree:       3
           / \
          0   4
           \
            2
           /
          1

low = 1, high = 3
```

Output:

```
Tree:       3
           /
          2
         /
        1
```

🧩 Time & Space Complexity:

Complexity	Explanation
⏱️ Time	O(n) — We may visit all nodes once
🧠 Space	O(h) — Due to recursion stack (`h` = height of tree)

✅ Verdict:

- Efficient approach using in-place pointer adjustments
- No need to recreate tree
- Suitable for large trees (up to 10^4 nodes)

✅ LeetCode 669 – Trim a Binary Search Tree (Hinglish)

🔍 Problem Samajh:

Tujhe ek **Binary Search Tree (BST)** diya gaya hai aur do values `low` aur `high` di gayi hain.

Tera kaam hai tree ko aise **trim** karna ki sirf wahi nodes bache jinka value `low` aur `high` ke beech ho, i.e., **[low, high]** range mein ho.

- Tree ka **relative structure** same rehna chahiye.
- Final tree ka **root change bhi ho sakta hai**.
- Return karna hai **new root**.

🧠 BST Property Ka Use:

- **Left subtree** → values chhoti hoti hain root se
- **Right subtree** → values badi hoti hain root se

Is property ka use karke hum unwanted nodes ko hata sakte hain.

🔧 Approach (If-Else Based):

✅ Step by Step:

1. Ek **dummy node** banate hain jiska left pointer root ko point karta hai (taaki root bhi agar out of range ho toh handle ho jaaye).
2. Left subtree ko trim karte hain:
    - Jab tak `root->left` exist karta hai:
        - Agar left ka value `low` se chhota hai → `root->left = root->left->right`
        - Agar left ka value `high` se bada hai → `root->left = root->left->left`
        - Agar value range mein ho → break

3. Right subtree ke liye bhi same logic lagate hain:
    - Jab tak `root→right` exist karta hai:
        - Agar value `high` se bada hai → `root→right = root→right→left`
        - Agar value `low` se chhota hai → `root→right = root→right→right`
        - Agar value range mein ho → break
4. Fir recursively left aur right ko trim karte hain.

👨‍💻 Code (C++ – If-Else Explanation Style):

```
class Solution {
public:
    void trim(TreeNode* root, int lo, int hi) {
        if (root == NULL) return;

        // LEFT ko trim karna
        while (root→left != NULL) {
            if (root→left→val < lo) {
                // left value chhoti hai, uska right bacha sakta hai
                root→left = root→left→right;
            } else if (root→left→val > hi) {
                // left value badi hai, uska left hi valid ho sakta hai
                root→left = root→left→left;
            } else {
                break; // value range mein hai
            }
        }

        // RIGHT ko trim karna
        while (root→right != NULL) {
            if (root→right→val > hi) {
                root→right = root→right→left;
            } else if (root→right→val < lo) {
                root→right = root→right→right;
            } else {
                break;
            }
        }

        // Ab bach gaya subtree, usko recursively trim karo
        trim(root→left, lo, hi);
        trim(root→right, lo, hi);
    }

    TreeNode* trimBST(TreeNode* root, int lo, int hi) {
        TreeNode* dummy = new TreeNode(INT_MAX); // dummy node banayi
        dummy→left = root;

        trim(dummy, lo, hi); // dummy se trim start kiya

        return dummy→left; // actual trimmed root return kiya
    }
};
```

🧪 Example:

Input:

```
Tree =      3
        / \
       0   4
        \
         2
        /
       1

low = 1, high = 3
```

Output:

```
Trimmed Tree =    3
               /
              2
             /
            1
```

🧠 Time & Space Complexity:

Complexity	Explanation
🕒 Time	O(n) – Saare nodes ko ek baar visit kar sakte ho
🧠 Space	O(h) – h = tree ka height (recursive stack)

🔑 Highlights:

- Dummy node se root handling simple ho jata hai.
- While loops se efficient pointer shifting hoti hai.
- Naaya tree banane ki zarurat nahi padti – **in-place trimming** hoti hai.