PW SKILLS

Lesson Plan

BST-3

PW SKILLS

Prerequisites:

- Understanding of basic data structures like arrays and linked lists.
- Understanding of recursion.
- Proficiency in a C++ programming language

Today's Checklist:

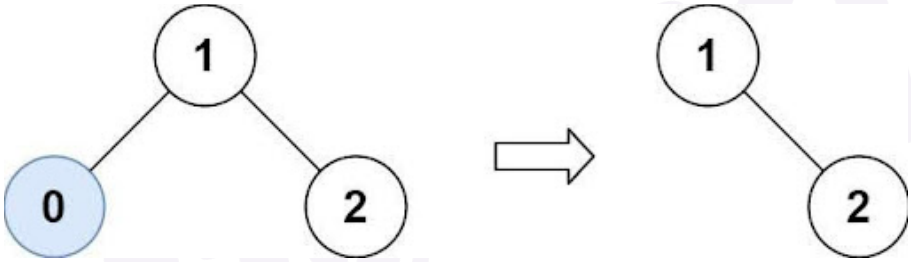
- Trim a BST [Leetcode-669]
- Morris Traversal = Inorder Traversal
- Flatten Binary Tree to Linked List [Leetcode-114]

Trim a Binary Search Tree [Leetcode-669]

Given the root of a binary search tree and the lowest and highest boundaries as low and high, trim the tree so that all its elements lies in [low, high]. Trimming the tree should not change the relative structure of the elements that will remain in the tree (i.e., any node's descendant should remain a descendant). It can be proven that there is a unique answer.

Return the root of the trimmed binary search tree. Note that the root may change depending on the given bounds.

Example 1:



Input: root = [1,0,2], low = 1, high = 2
Output: [1,null,2]

Code:

```
class Solution {
public:
    TreeNode* trimBST(TreeNode* root, int low, int high) {
        if (!root) {
            // checking if the root is null or not.
            return root;
        }

        if (root->val >= low && root->val <= high) {
            // checking root's val should be equal or inside the low and
high.
            root->left = trimBST(root->left, low, high);
            root->right = trimBST(root->right, low, high);
            return root;
        }
```

```
        if (root->val < low) {
            // return right subtree if the root is less than low.
            return trimBST(root->right, low, high);
        }

        // return left subtree if the root is greater than high.
        return trimBST(root->left, low, high);
    }
};
```

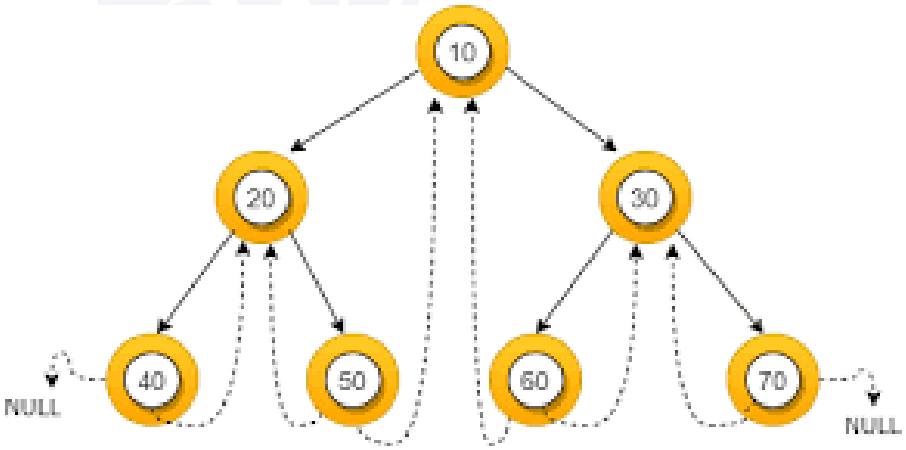
Explanation:

- The trimBST function recursively trims a binary search tree (BST) based on given low and high values.
- If the root is null, it returns null.
- If the root's value is within the range [low, high], it recursively trims both left and right subtrees and returns the modified root.
- If the root's value is less than low, it trims the left subtree and returns the result.
- If the root's value is greater than high, it trims the right subtree and returns the result.
- The goal is to prune the tree such that all node values are within the specified range [low, high].

Time Complexity: O(N), where N is the number of nodes in the binary search tree. In the worst case, the algorithm visits all nodes once.

Space Complexity: O(H), where H is the height of the binary search tree. The space complexity is dominated by the recursive call stack, and in the worst case (unbalanced tree), the height of the stack is H. In the average case, for a balanced tree, the space complexity is O(log N).

Morris Traversal = Inorder Traversal



Using Morris Traversal, we can traverse the tree without using stack and recursion. The idea of Morris Traversal is based on Threaded Binary Tree. In this traversal, we first create links to Inorder successor and print the data using these links, and finally revert the changes to restore original tree.

Code:

```
#include <bits/stdc++.h>
using namespace std;

/* A binary tree tNode has data, a pointer to left child
   and a pointer to right child */
struct tNode {
    int data;
    struct tNode* left;
    struct tNode* right;
};

/* Function to traverse the binary tree without recursion
   and without stack */
void MorrisTraversal(struct tNode* root)
{
    struct tNode *current, *pre;

    if (root == NULL)
        return;

    current = root;
    while (current != NULL) {

        if (current->left == NULL) {
            cout << current->data << " ";
            current = current->right;
        }
        else {

            /* Find the inorder predecessor of current */
            pre = current->left;
            while (pre->right != NULL
                   && pre->right != current)
                pre = pre->right;

            /* Make current as the right child of its
               inorder predecessor */
            if (pre->right == NULL) {
                pre->right = current;
                current = current->left;
            }

            else {
                pre->right = NULL;
                cout << current->data << " ";
                current = current->right;
            } /* End of if condition pre->right == NULL */
        } /* End of if condition current->left == NULL*/
    } /* End of while */
}
```

```
struct tNode* newtNode(int data)
{
    struct tNode* node = new tNode;
    node→data = data;
    node→left = NULL;
    node→right = NULL;

    return (node);
}
```

Explanation:

- The code implements Morris Traversal, a tree traversal method without recursion or a stack.
- It defines a binary tree node structure (tNode) with data, left, and right pointers.
- The MorrisTraversal function takes the root of a binary tree as input and prints its elements in inorder.
- It uses threaded binary tree approach, modifying the tree structure temporarily for traversal.
- The traversal process avoids using a stack by utilizing the right pointers of certain nodes.
- The algorithm efficiently finds and utilizes the inorder predecessor of each node during traversal.
- The code includes utility functions to create a new tree node (newtNode).
- Overall, Morris Traversal reduces space complexity compared to traditional recursive or stack-based approaches.

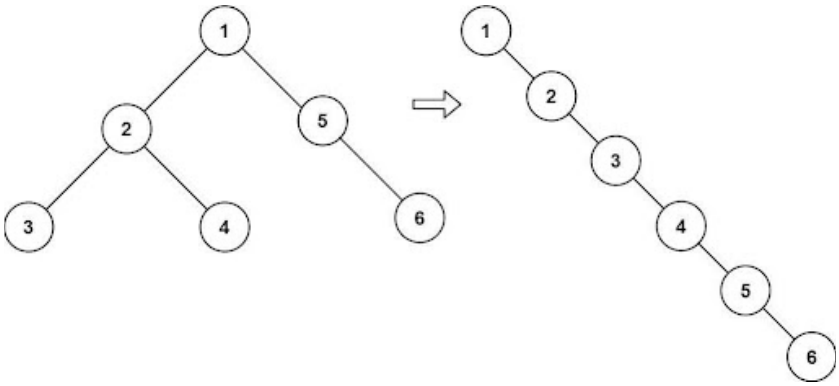
Time Complexity:
O(n) - The traversal visits each node exactly once, making it a linear time complexity.

Space Complexity:
O(1) - Morris Traversal is an in-place traversal method, meaning it does not require additional space proportional to the input size. It achieves constant space complexity.

Flatten Binary Tree to Linked List [Leetcode-114]

Given the root of a binary tree, flatten the tree into a "linked list":
The "linked list" should use the same TreeNode class where the right child pointer points to the next node in the list and the left child pointer is always null.
The "linked list" should be in the same order as a pre-order traversal of the binary tree.

Example 1:



Input: root = [1,2,5,3,4,null,6]
Output: [1,null,2,null,3,null,4,null,5,null,6]

Code:

```
class Solution {
public:
    void traversal (vector<int>& v, TreeNode* node){
        if(node == NULL) return;
        v.push_back(node->val);
        traversal(v,node->left);
        traversal(v,node->right);
    }

    void flatten(TreeNode* root) {
        vector<int> v;
        traversal(v, root);
        TreeNode* head = new TreeNode();
        TreeNode* cursor = head;
        for(int i=0; i<v.size(); i++ ){
            TreeNode* temp = new TreeNode(v[i]);
            cursor->right = temp;
            cursor = cursor->right;
        }

        cursor = head->right;
        TreeNode* cursor1 = root;

        while(cursor){
            cursor1->val = cursor->val;
            cursor1->left = NULL;
            cursor1->right = cursor->right;
            cursor = cursor->right; cursor1 = cursor1->right;
        }


    }
};
```

Explanation:

- The traversal function performs a pre-order traversal on a binary tree, storing node values in a vector.
- The flatten function flattens a binary tree to a right-skewed linked list using the pre-order traversal result.
- It creates a new linked list using TreeNode instances and copies values from the vector to the linked list.
- The original tree is modified to represent the flattened structure by updating node values and adjusting left and right pointers.
- The cursor1 and cursor pointers are used to navigate the original tree and the newly created linked list, respectively.

Time Complexity:
O(N), where N is the number of nodes in the binary tree. The pre-order traversal visits each node once, and the subsequent creation of the linked list also takes linear time.

Space Complexity:
O(N), where N is the number of nodes in the binary tree. The space complexity is dominated by the auxiliary vector used to store node values during traversal. Additionally, the recursive call stack during traversal contributes to the space complexity. The linked list created also has O(N) space complexity since each node in the tree is represented in the linked list.

PW SKILLS

THANK YOU !

PW SKILLS