PW SKILLS

Lesson Plan

BST-3

Java

PW SKILLS

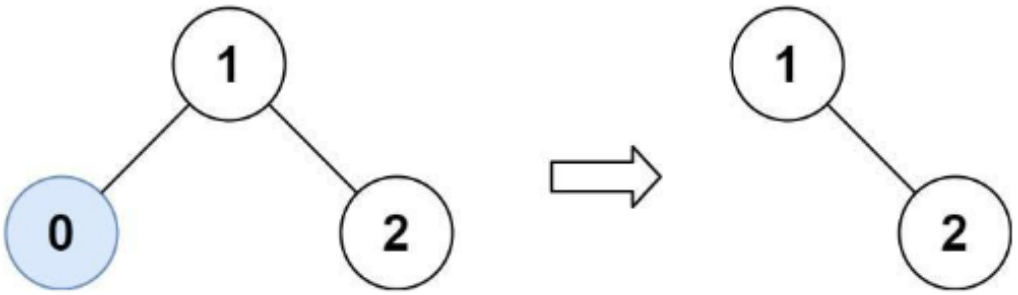
Today's Checklist

- Trim a BST
- Morris Traversal
- Flatten Binary Tree to Linked List

Q. 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.



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

Code:

```
public class Solution {
    public TreeNode trimBST(TreeNode root, int low, int high) {
        if (root == null) return null;

        if (root.val < low) {
            return trimBST(root.right, low, high);
        } else if (root.val > high) {
            return trimBST(root.left, low, high);
        }

        root.left = trimBST(root.left, low, high);
        root.right = trimBST(root.right, low, high);

        return root;
    }
}
```

Explanation: If the root->val is lower than the range, then return the right node because all nodes to the right are higher.
If the root->val is higher than the range, then return the left node because all nodes to the left are lower.
then recurse on node->left and node->right.

Time Complexity: O(n) - Visits each node once, checking and potentially trimming the tree based on the given range.

Space Complexity: O(h) - Due to recursive calls, occupying space in the call stack proportional to the tree's height.

Morris 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.

1. Initialize current as root
2. While current is not NULL

If the current does not have left child

a) Print current's data
b) Go to the right, i.e., current = current->right

Else

a) Find rightmost node in current left subtree OR
node whose right child == current.

If we found right child == current

a) Update the right child as NULL of that node whose right child is current
b) Print current's data
c) Go to the right, i.e. current = current->right

Else

a) Make current as the right child of that rightmost node we found; and
b) Go to this left child, i.e., current = current->left

Although the tree is modified through the traversal, it is reverted back to its original shape after the completion. Unlike Stack based traversal, no extra space is required for this traversal.

Code:

```
class TNode {
    int data;
    TNode left, right;

    TNode(int item) {
        data = item;
        left = right = null;
    }
}

public class BinaryTree {
    TNode root;
```

```
    void morrisTraversal(TNode root) {
        TNode current, pre;

        if (root == null)
            return;

        current = root;
        while (current != null) {
            if (current.left == null) {
                System.out.print(current.data + " ");
                current = current.right;
            } else {
                pre = current.left;
                while (pre.right != null && pre.right != current)
                    pre = pre.right;

                if (pre.right == null) {
                    pre.right = current;
                    current = current.left;
                } else {
                    pre.right = null;
                    System.out.print(current.data + " ");
                    current = current.right;
                }
            }
        }
    }

    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        tree.root = new TNode(1);
        tree.root.left = new TNode(2);
        tree.root.right = new TNode(3);
        tree.root.left.left = new TNode(4);
        tree.root.left.right = new TNode(5);

        tree.morrisTraversal(tree.root);
    }
}
```

Output: 4 2 5 1 3

Time Complexity: O(n) If we take a closer look, we can notice that every edge of the tree is traversed at most three times. And in the worst case, the same number of extra edges (as input tree) are created and removed.

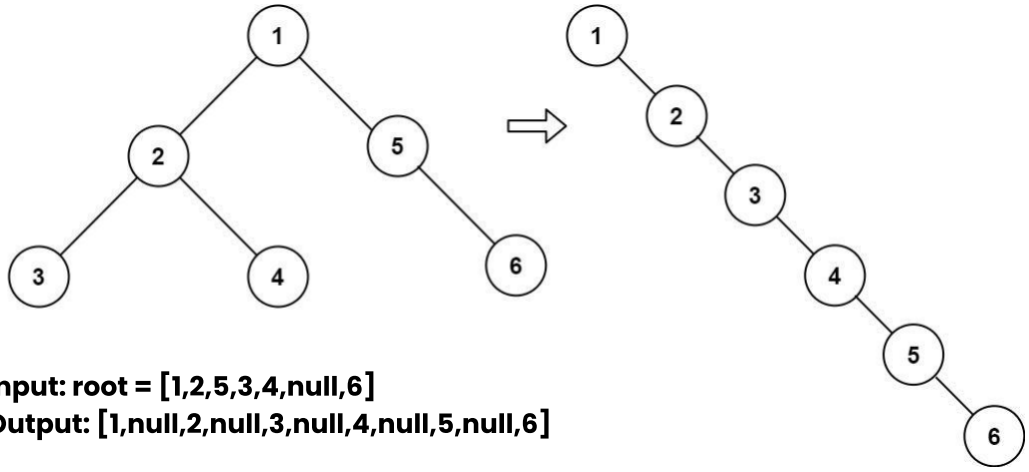
Auxiliary Space: O(1) since using only constant variables.

Q. 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.



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

Code:

```
class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int value) {
        val = value;
        left = right = null;
    }
}

public class Solution {

    public TreeNode rightmost(TreeNode root) {
        if (root.right == null) return root;
        return rightmost(root.right);
    }

    public void flatten(TreeNode root) {
        if (root == null) return;
        TreeNode nextright;
        TreeNode rightMOST;

        while (root != null) {

            if (root.left != null) {
                rightMOST = rightmost(root.left);
                nextright = root.right;
                root.right = root.left;
                root.left = null;
                rightMOST.right = nextright;
            }
            root = root.right;
        }
    }
}
```

Explanation: APPROACH -> O (N) SPACE
By using brute force, we can traverse the tree in preorder manner and store the result in a vector.
Later using that vector, we can arrange nodes, such as

vector[i] -> right = vector[i+1];
vector[i] -> left = NULL;
APPROACH -> O (1) SPACE

From the diagram given , it can be seen all nodes are present on the right.
All the nodes in left subtree come before the nodes in right subtree.
For each node i

If there is no left node -> move to next right node.
If left is present ->
Store the right subtree
ADD left subtree to right of root,
Now add the stored right subtree to the rightmost node of current tree.
Also make node -> left =NULL.

Time Complexity: O(n) - where n is the number of nodes in the binary tree. The function traverses each node once, performing constant time operations within each iteration.

Space Complexity: O(1) - The function doesn't use any additional space that scales with the input. It operates in constant space regardless of the size of the tree, utilizing only a few auxiliary pointers.

PW SKILLS

THANK YOU !

PW SKILLS