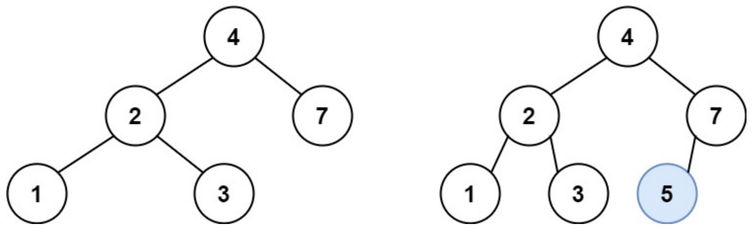
Day 36 / 100 :

Topic - Tree, Linked list

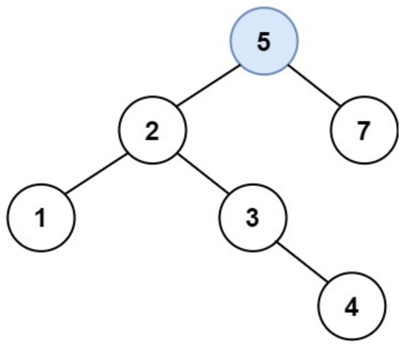
1 Problem statement: Insert Into Binary Tree (Medium)

You are given the root node of a binary search tree (BST) and a value to insert into the tree. Return the root node of the BST after the insertion. It is guaranteed that the new value does not exist in the original BST.
Notice that there may exist multiple valid ways for the insertion, as long as the tree remains a BST after insertion. You can return any of them.

Example 1:



Input: root = [4,2,7,1,3], val = 5
Output: [4,2,7,1,3,5]
Explanation: Another accepted tree is:



Example 2:
Input: root = [40,20,60,10,30,50,70], val = 25
Output: [40,20,60,10,30,50,70,null,null,25]

Example 3:
Input: root = [4,2,7,1,3,null,null,null,null,null,null], val = 5
Output: [4,2,7,1,3,5]

Solutions :

Approach 1 - Binary Tree

If the root is empty, the new tree node can be returned as the root node.
Otherwise compare root. val is related to the size of the target value:

- If root. val is greater than the target value, indicating that the target value should be inserted into the left subtree of the root.For thet we have to reach to the leftmost node of the left subtree.
- If root. val is lesser than the target value, indicating that the target value should be inserted into the right subtree of the root.For thet we have to reach to the rightmost node of the right subtree.

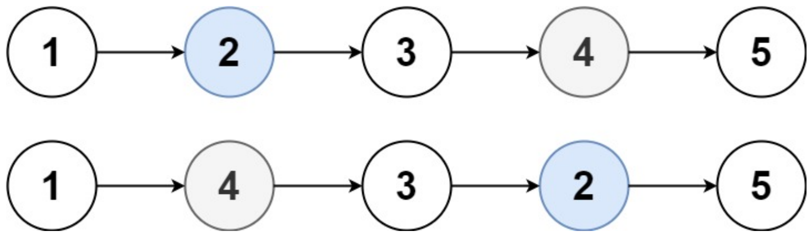
```
class Solution {
public:
    TreeNode* insertIntoBST(TreeNode* root, int val) {
        if(!root) return new TreeNode(val);
        TreeNode* temp=root;
        while(true){
            if(val<=temp->val){
                if(temp->left!=NULL){
                    temp=temp->left;
                }
                else{
                    temp->left=new TreeNode(val);
                    break;
                }
            }
            else{
                if(temp->right!=NULL){
                    temp=temp->right;
                }
                else{
                    temp->right=new TreeNode(val);
                    break;
                }
            }
        }
        return root;
    }
};
```

Time Complexity : BigO(N), where N is height of binary search tree.
Space Complexity : BigO(1)

2 Problem statement: Swapping Node in a linked list (Medium)

You are given the head of a linked list, and an integer k.
Return the head of the linked list after swapping the values of the kth node from the beginning and the kth node from the end (the list is 1-indexed).

Example 1:



Input: head = [1,2,3,4,5], k = 2
Output: [1,4,3,2,5]

Example 2:
Input: head = [7,9,6,6,7,8,3,0,9,5], k = 5
Output: [7,9,6,6,8,7,3,0,9,5]

Solutions :

Approach 1 - Binary Tree

Intution:
Traverse the list twice - once to find the kth node from the beginning, and then again to find the kth node from the end. After the two nodes are identified, their values can be swapped.

Approach:

- First initializes two pointers left and right to point to the head of the list. It then moves the left pointer k-1 times to reach the k-th node from the beginning of the list. At this point, left points to the node that needs to be swapped.
- The function then initializes a new pointer curr to point to the same node as left. It then moves curr forward until it reaches the end of the list, while also moving right pointer to point to the kth node from the end of the list. Once curr reaches the end of the list, right will point to the kth node from the end.
- Finally, the swap the values of the two nodes pointed to by left and right, and returns the head of the modified list.

Complexity:

- Time complexity:O(n)
- Space complexity:O(1)

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* swapNodes(ListNode* head, int k) {
          ListNode *left = head, *right = head;
    for(int i = 1; i < k; i++)
        left = left->next;
    ListNode *curr = left;
    while(curr->next != nullptr){
        curr = curr->next;
        right = right->next;
    }
    int t = left->val;
    left->val = right->val;
    right->val = t;
    return head;
    }
};
```