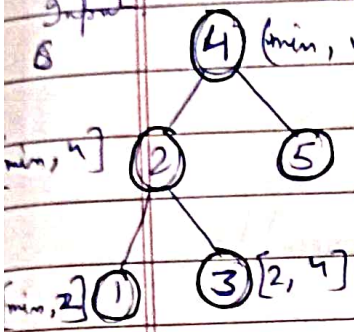
Lecture - 74

Interview Problems on BST

Q1: Check if it is a BST or Not.



1. Every node lies in its range.
2. Left Subtree is BST.
3. Right Subtree is BST.

T.C = O(N)
S.C = O(h)

Sol.⇒

```
class Solution {
public:
    bool isValidBSThelper(TreeNode* root, long minValue, long maxValue) {
        if (root == NULL) {
            return true;
        }
// 1. Root Node value should lie in its Range
        if (root->val <= minValue || root->val >= maxValue) {
            return false;
        }
// left Subtree
// Right Subtree
        return isValidBSThelper(root->left, minValue, root->val)
            && isValidBSThelper(root->right, root->val, maxValue);
    }
    bool isValidBST(TreeNode* root) {
        return isValidBSThelper(root, LONG_MIN, LONG_MAX);
    }
};
```

⇒ **Method 2 → Inorder of BST is Sorted**

T.C = S.C = O(N)

```
Class Solution {
public:
    bool isValidBSThelper(TreeNode* root, TreeNode* &prev) {

        if (root == NULL)
            return true;
        }
//left subtree    if (!isValidBSThelper(root->left, prev)) {
            return false;
        }
//Root Node       if (prev != NULL && root->val <= prev) {
            return false;
        }
        prev = root;
//Right subtree   return isValidBSThelper(root->right, prev);
    }
    bool isValidBST(TreeNode* root) {
        TreeNode* prev = NULL;
        return isValidBSThelper(root, prev);
    }
};
```

Q2) Given two vectors that represent a sequence of keys. Imagine we make a BST from each array. We need to tell Whether two BSTs will be identical or not without actually constructing the tree.

Input:- arr1 = {4, 2, 5, 1, 3}
arr2 = {4, 5, 2, 3, 1}

Output:- BSTs are identical.

① Find the 1st elements within range in both arrays
② if both elements are leaf nodes → return true
③ if one element is leaf node return false
④ if elements not same → false
⑤ recursively check for left & right subtrees

T.C → $O(N^2)$
S.C → $O(h)$

Date
Page

```
#include<vector>
using namespace std;

bool checkIdenticalBSThelper(vector<int>&V1, vector<int>&V2
                             int a1, int a2, int minValue,
// find 1st element in V1 within range   int maxValue){
    int i;
    for(i=a1; i<V1.size(); i++){
        if(V1[i]>minValue && V1[i]<maxValue) break;
    }
// find 1st element in V2 within range
    int j;
    for(j=a2; j<V2.size(); j++){
        if(V2[j]>minValue && V2[j]<maxValue) break;
    }
// if no element is find within range (leaf nodes)
    if(i=V1.size() && j=V2.size()) return true;  // base case
// if only one vector doesn't have element within range
    if(((i=V1.size() && j!=V2.size()) && (i!=V1.size() && j=V2.size()))
        return false;
//checking if both elements are equal
    if(V1[i]!=V2[j]) return false;
//recursively check for left and right subtree
    return checkIdenticalBSThelper(V1,V2,a1+1,a2+1,minValue,V1[i])
        && checkIdenticalBSThelper(V1,V2,a1+1,a2+1,V1[i],maxValue);
}
bool checkIdenticalBSTs(Vector<int>&V1, Vector<int>&V2){
    return checkIdenticalBSThelper(V1,V2,0,0,~~minValue~~
                                   INT_MIN, INT_MAX);
}
int main(){
    Cout<<"Enter input-"<<endl;
```

FREEMIND

Date

Page

```
int n; cin>>n;
Vector<int> V1(n), V2(n);

for(int i=0; i<n; i++){
    cin>>V1[i];
}
for(int i=0; i<n; i++){
    cin>>V2[i];
}

if(checkIdenticalBSTs(V1,V2)){
        cout<<"BSTs are Identical"<<endl;
}else{
        cout<<"BSTs are Not identical;
}
return 0;
}
```

Q3→ Given the preorder traversal of a BST, construct the BST.

Sol.1

```
Class Solution{
public:
    TreeNode *bstFromPreorderHelper(Vector<int>& preorder, int prestart,
            int preend, vector<int>& inorder, int instart,
            int inend, Unordered_map<int,int> inmap){
    if(prestart>preend || instart>inend) return NULL;
        TreeNode* root = new TreeNode(preorder[prestart]);
        int inroot_idx = inmap[preorder[prestart]];
        int leftsubtreeElements = inroot_idx - instart;

        root->left = bstFromPreorderHelper(preorder, prestart+1,
            prestart+leftsubtree_elements, inorder, instart,
                inroot_idx-1, inmap);
```

```
    root->right = bstFrom_preorder_helper(preorder, prestart +
                  leftsubtree_elements+1, preend, inorder,
                  inroot_idx+1, inend, inmap);
    return root;
}

TreeNode* bstFrompreorder(Vector<int>& preorder) {
    Vector<int> inorder = preorder;
    sort(inorder.begin(), inorder.end());

    Unordered_map<int, int> inmap;
    for(int i = 0;i<inorder.size(); i++) {
        inmap[inorder[i]] = i; // Storing value index pairs
    }
    return bstFrompreorderhelper(preorder, 0, preorder.size()-1,
                  inorder, 0, inorder.size()-1, inmap);
    }
};
```

T.C → O(n log n) + O(n)
S.C → O(n) + O(n) + O(h)

Method 2s

```
class Solution {
public:
    TreeNode* bstFrompreorderhelper(Vector<int>& preorder, int& index,
                                    int upperbound) {
        if(index >= preorder.size()) {
            return NULL;
        }
        if(preorder[index] >= upperbound) {
            return NULL;
        }
        TreeNode* root = new TreeNode(preorder[index]);
        index++;
```

```
    root->left = bstFromPreorderHelper(preorder, index, root->val);
    root->right = bstFromPreorderHelper(preorder, index, upperbound);

    return root;
  }
  TreeNode* bstFromPreorder(Vector<int>& preorder) {
    int index = 0;
    return bstFromPreorderHelper(preorder, index, INT_MAX);
  }
};
```

Q4 ⇒ Given a binary tree. Write a function that returns the size of the largest subtree which is also a Binary search tree (BST). If the complete Binary Tree is BST, then return the size of the whole tree.

Q5 ⇒ Given a binary tree root, return the maximum sum of all keys of any sub-Tree which is also a Binary Search Tree (BST).

Ans ⇒

```
Class NodeInfo {
  public:
    int minValue;
    int maxValue;
    int maxCurrentSum;

    NodeInfo(int min, int max, int sum)
        minValue = min;
        maxValue = max;
        maxCurrentSum = sum;
    }
}
```

$T.C = S.C = O(n)$

```
class Solution {
public:
    int ans = 0;
    NodeInfo maxSumBSTHelper(TreeNode * root) {

        if (!root) {
            return NodeInfo(INT_MAX, INT_MIN, 0);
        }
        NodeInfo leftSubtree = maxSumBSTHelper(root->left);
        NodeInfo rightSubtree = maxSumBSTHelper(root->right);

        if (root->val > leftSubtree.maxValue && root->val <
                rightSubtree.minValue) {
            //root node forms a BST
            ans = max(ans, leftSubtree.maxCurrentSum +
                           rightSubtree.maxCurrentSum +
                           root->val);
            return NodeInfo(min(leftSubtree.minValue, root->val),
                            max(rightSubtree.maxValue, root->val),
leftSubtree.maxCurrentSum + rightSubtree.maxCurrentSum + root->value);

        }
        //root node does not form a BST
        return (INT_MIN, INT_MAX, max(leftSubtree.
                                      maxCurrentSum,
                                      rightSubtree.maxCurrentSum));
    }
    int maxSumBST(Treenode * root) {
        maxSumBSTHelper(root);
        return ans;
    }
};
```

FREEMIND

Date

Page

Q6) Write a C++ program to construct all unique binary search trees (BSTs) for keys ranging from 1 to N. The program should prompt the user to enter the value of N, and then construct and display all possible BSTs for the given range of keys.

Sol)

from $i = 1$ to N

i = root Node

left_Subtrees = $f(1, i-1)$

right_Subtrees = $f(i+1, N)$

// watch every leftSubree with every right Subtree with i as root Node

```
Class Solution{
public:
    Vector<TreeNode*> generateTreesHelper(int start, int end){
        Vector<TreeNode*> treeList;
        //base case
        if(start>end){
            treelist.push_back(NULL);
            return treeList;
        }
        for(int i=start; i<=end; i++){

            //recursive case
            Vector<TreeNode*> left_Subtrees =
                                    generateTreesHelper(start,i-
            Vector<TreeNode*> right_Subtrees =
                                    generateTreesHelper(i+1, end);
```

```
        for (TreeNode* leftSubtree : leftSubtrees) {
            for (TreeNode* rightSubtree : rightSubtrees) {
                TreesNode* rootNode = new TreeNode(i);
                rootNode -> left = leftSubtree;
                rootNode -> right = rightSubtree;
                treesList.push_back(rootNode);
            }
        }
    }
    return treesList;
}
Vector<TreeNode*> generateTrees(int n) {
    return generateTreesHelper(1, n);
}
};
```

Q7> Given an array of integers, replace every element with the least greater element on its right side in the array. If there are no greater elements on the right side, replace it with -1.

Input:- 8, 3, 10, 1, 6, 9, 14

Output:- 9, 6, 14, 6, 9, 14, -1

① Brute Force
→ Nested Loops
→ $O(N^2)$

② BST
Inorder Successor

$O(h \log n)$
$O(N^2)$ (worst case)

FREEMIND

Date

Page

~~Lecture 35~~ (~~Generic Trees~~)

~~Q) What are generic trees~~

```
#include <iostream>
#include <vector>
using namespace std;

Class TreeNode {
    public:
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode (int v) {
        val = v;
        left = right = NULL;
    }
};

TreeNode* insertNode ( TreeNode *root, int element,
                       int &Successor) {
    if (root == NULL) {
        return new TreeNode (element);
    }
    if(element < root->val) {
        successor = root->val;
        root->left = insertNode(root->left, element,
                                successor);
    }
    else if (element > root->val) {
        root->right = insertNode(root->right,
                                 element, successor);
    }
    return root;
}
```

```
Void replaceWithLeastGreaterElement(vector<int> &v){
    TreeNode* root = NULL;

    for( int i = v.size()-1; i>=0 ; i--) {
        int successor = -1;
        root = insertNode(root, v[i], successor);
        v[i] = successor;
    }
}
int main() {
    cout<<"Enter Input:" <<endl;
    int n;
    cin>> n;
    Vector<int> V(n);
    for(int i=0; i<n; i++) {
        cin>> V[i];
    }
    replaceWithLeastGreaterElement(V);
    for(int i=0; i<n; i++) {
        cout<< V[i] <<" ";
    } cout<<endl;
    return 0;
}
```