Nevil Ladani
npl130030@utdallas.edu

Algorithms - Project Challenge

Largest Binary Search Tree in a given Binary Tree.

Project Description:

Given a binary tree (each node contains a positive integer key), find the largest subtree that is a Binary Search Tree (i.e, the nodes satisfy BST property).

Input:

The first line contains an integer D, the number of data sets to follow. Each data set consists of three lines. The first line of each data set has an integer N which is the number of of nodes in the tree. The second line in the data set has N integers in the form of the post order traversal of the tree. The third line in the data set has N integers in the form of the in order traversal of the tree.

Output:

For each test case, output the number of nodes in the largest BST.

Approach

- Using the post order and in order traversal of the tree we make the binary tree.

- Once the tree is formed for all nodes of the tree taking one node at a time as current node , check for the largest binary search tree with the current node as root node.

- Output the root node and number of nodes of the largest binary search tree along with all its nodes.

Algorithms:

Algorithm for making tree: - bTree(inorder, postorder)

Input: Inorder, Postorder

Output: Binary Tree

```
bTree (inorder,  postorder)
        return addTree(inorder, 0, inorder.length - 1, postorder, 0, postorder.length - 1)

addTree(inorder, int is, int ie, postorder, int ps, int pe)
{
        if (is > ie OR ps > pe) then
                return null
        rootVal = postorder[pe]
        TreeNode root =  TreeNode(rootVal)
```

```
for (i = is to ie) do
{
        if (inorder[i] = rootVal) then
                TreeNode left = addTree(inorder, is, i - 1, postorder, ps, ps + i - is - 1);
                TreeNode right = addTree(inorder, i + 1, ie, postorder, pe - ie + i, pe - 1);
                if (right!=null) then
                        right.parent = root;
                if (left!=null)
                        left.parent = root;
                root.maxSubVal = 1
                root.left = left;
                root.right = right;
}
```

Algorithm for getting maximum Binary Search Tree

Algorithm for making tree: - maxBST

Input: maxBST(Tree)

Output: Maximum Binary Search Tree

maxBST()

```
        t = buildTree (inorder,postorder)
        searchMax(t.node,0,∞, t.node,0) // find max binary search tree for the root node of tree t.

        TreeNode finalNode = maxSub(t.node, t.node) //gives root node of maximum binary search tree.

        Print ("Root node for maximum sub-BST: "+finalNode.val) // root node of maximum BST
        Print ("Number of Nodes in maximum sub-BST: "+finalNode.maxSubVal)
        Print ("Preorder traversal - Tree Structure")
        searchMax(finalNode,0, ∞, finalNode,1) // if the flag(last parameter) is set to 1 then the
                search is used to print all the nodes considered node.
```

TreeNode maxSub(TreeNode rNode, TreeNode maxN)
// to get maximum binary search tree at root node rNode and to return the root node with largest binary search tree(maxN).

```
{

        TreeNode rootNode = rNode;
        TreeNode maxNode = maxN;

        searchMax(rootNode.left,0,∞,rootNode.left,0);
        searchMax(rootNode.right,0, ∞,rootNode.right,0);

        if(maxNode.maxSubVal<rootNode.left.maxSubVal)
                maxNode = rootNode.left;
        else if(maxNode.maxSubVal<rootNode.right.maxSubVal)
                maxNode = rootNode.right;
        if(rootNode.left!=null)
                maxSub(rootNode.left, maxNode)
        if(rootNode.right!=null)
                maxSub(rootNode.right, maxNode)

        return maxNode;
}
```

Proof of correctness for maxSub.

Base Case: There is only one node i.e. n=1. The resulting maximum binary search tree would be that node.

Inductive Hypothesis: For all the node in the tree with depth less than k we have to keep track of a particular root node with maximum binary search tree.

Inductive Step: For any node with depth k. We find maximum of the BST out of BST for new node as root node and the previous max BST. We propagate the node with maximum BST down the tree to finally get the maximum BST.

searchMax(TreeNode tn, int mlV, int mrV, TreeNode tc,int flag) // to get the maximum binary search tree at node tc.

```
// if the flag is set to 1 then the search is used to print all the nodes.
{

        int maxlV = mlV // to keep track to the highest possible value in left subtree.
        int maxrV = mrV // to keep track to the lowest possible value in right subtree.

        if(flag=1)
                Print(tn.val)

        if((tn.left.val< tn.val) AND (tn.left.val >maxlV))
        {
                if(tn.parent.val < tn.val)
                        maxlV = tn.parent.val
                if(tn.left.val > maxlV)
                        tc.maxSubVal++  // increase the no. of nodes by one for BST withroot node tc.
                        searchMax(tn.left, maxlV,maxrV, tc,flag)
        }
        if( (tn.right.val> tn.val) AND (tn.right.val <maxrV))
        {
                if(tn.parent.val > tn.val)
                        maxrV = tn.parent.val
                if(tn.right.val <maxrV)
                        tc.maxSubVal ++ // increase the no. of nodes by one for BST with root node tc.
                        searchMax(tn.right, maxlV,maxrV, tc,flag)
        }
}
```

Proof of Correctness for searchMax:

Base Case: For n=1. The searchMax() gives the single node as maximum binary search tree.

Inductive Hypothesis: We have the binary search tree for a node x and height k. We also have all its corresponding nodes.

Inductive Step: For all the leaf nodes of the binary tree with root x we check the corresponding left and right node for its compatibility to form binary search tree. If there are any the height of the binary search tree would increase to k+1. The process will continue until there are no further compatible nodes. It would keep track of total number of nodes in binary search tree in variable maxSubVal of the current root node.

Time Complexity for searchMax()

n is the number of nodes in the tree.
T(n) = O(n)

Time Complexity for maxBST.

n is the number of nodes in the tree.
T(n) = Time complexity for searchMax X n + Time complexity of bTree (i.e.making tree)
T(n) = O(n^2) + O(n)
T(n) = O(n^2)

Program

We have used java to program the solution.
We have used eclipse as our IDE.

All the classes used:
maxBST.java - Main class
TreeNode.java - class to define tree node.
buildTree.java - class to sort build tree from given post order and in order traversals.