Build bst from scratch

class Node {

    int key;

    Node left, right;

    public **Node**(int *item*) {

        key = *item*;

        left = right = null;

    }

}

class BST {

    Node root;

    public **BST**() {

        root = null;

    }

    void **insert**(int *key*) {

        root = **insertH**(root, *key*);

    }

    Node **insertH**(Node *root*, int *key*) {

        if (*root* == null) {

*root* = new **Node**(*key*);

            return *root*;

        }

        if (*key* < *root*.key)

*root*.left = **insertH**(*root*.left, *key*);

        else if (*key* > *root*.key)

*root*.right = **insertH**(*root*.right, *key*);

        return *root*;

    }

    void **delete**(int *key*) {

        root = **deleteH**(root, *key*);

    }

    Node **deleteH**(Node *root*, int *key*) {

        if (*root* == null)

            return *root*;

        if (*key* < *root*.key)

*root*.left = **deleteH**(*root*.left, *key*);

        else if (*key* > *root*.key)

*root*.right = **deleteH**(*root*.right, *key*);

        else {

            if (*root*.left == null)

                return *root*.right;

            else if (*root*.right == null)

                return *root*.left;

*root*.key = **minValue**(*root*.right);

*root*.right = **deleteH**(*root*.right, *root*.key);

        }

        return *root*;

    }

    int **minValue**(Node *root*) {

        int minv = *root*.key;

        while (*root*.left != null) {

            minv = *root*.left.key;

*root* = *root*.left;

        }

        return minv;

    }

    void **inorder**() {

**inorderRec**(root);

        System.out.**println**("\n");

    }

    void **inorderRec**(Node *root*) {

        if (*root* != null) {

**inorderRec**(*root*.left);

            System.out.**print**(*root*.key + " ");

**inorderRec**(*root*.right);

        }

    }

}

public class Main{

    public static void **main**(String[] *args*) {

        BST tree = new **BST**();

        tree.**insert**(50);

        tree.**insert**(30);

        tree.**insert**(20);

        tree.**insert**(40);

        tree.**insert**(70);

        tree.**insert**(60);

        tree.**insert**(80);

        System.out.**println**("Inorder traversal:");

        tree.**inorder**();

*// tree.delete(20);*

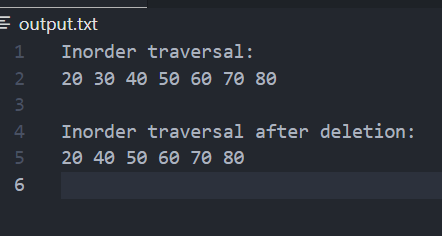
        tree.**delete**(30);

        System.out.**println**("Inorder traversal after deletion:");

        tree.**inorder**();

    }

}

OUTPUT:  


2. Validate bst

import java.util.\*;

class Node {

    int data;

    Node left, right;

    public **Node**(int *data*) {

        this.data = *data*;

        left = right = null;

    }

}

class BT {

    Node root;

    public **BT**() {

        root = null;

    }

    void **constructBT**(Integer[] *arr*) {

        if (*arr* == null || *arr*.length == 0 || *arr*[0] == null)

            return;

        root = new **Node**(*arr*[0]);

        Queue<Node> queue = new **LinkedList**<>();

        queue.**add**(root);

        int i = 1;

        while (i < *arr*.length) {

            Node current = queue.**poll**();

            if (i < *arr*.length && *arr*[i] != null) {

                current.left = new **Node**(*arr*[i]);

                queue.**add**(current.left);

            }

            i++;

            if (i < *arr*.length && *arr*[i] != null) {

                current.right = new **Node**(*arr*[i]);

                queue.**add**(current.right);

            }

            i++;

        }

    }

    boolean **validateBST**(Node *node*, Node *minVal*, Node *maxVal*) {

        if (*node* == null)

            return true;

        if ((*minVal* != null && *node*.data <= *minVal*.data) || (*maxVal* != null && *node*.data >= *maxVal*.data))

            return false;

        return **validateBST**(*node*.left, *minVal*, *node*) && **validateBST**(*node*.right, *node*, *maxVal*);

    }

    void **inorder**(Node *node*) {

        if (*node* != null) {

**inorder**(*node*.left);

            System.out.**print**(*node*.data + " ");

**inorder**(*node*.right);

        }

    }

    void **display**() {

**inorder**(root);

        System.out.**println**();

    }

}

public class Main {

    public static void **main**(String[] *args*) {

        Scanner sc = new **Scanner**(System.in);

*// Taking input as an array*

        int n = sc.**nextInt**();

        Integer[] arr = new Integer[n];

        for (int i = 0; i < n; i++) {

            String input = sc.**next**();

            arr[i] = input.**equals**("null") ? null : Integer.**parseInt**(input);

        }

        BT tree = new **BT**();

        tree.**constructBT**(arr);

        System.out.**println**("Inorder Traversal of the Tree:");

        tree.**display**();

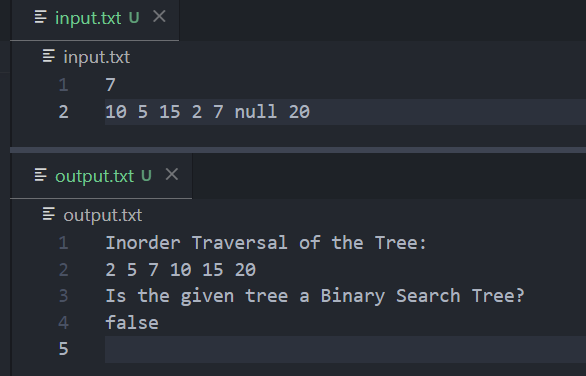
        tree.root.data = 1;

        System.out.**println**("Is the given tree a Binary Search Tree?");

        System.out.**println**(tree.**validateBST**(tree.root, null, null));

    }

}

OUTPUT:  


4 views of binary tree

import java.util.\*;

class Node {

    int data;

    Node left, right;

    public **Node**(int *data*) {

        this.data = *data*;

        this.left = null;

        this.right = null;

    }

}

class BinaryTree {

    Node root;

    public **BinaryTree**() {

        this.root = null;

    }

    public void **constructBT**(Integer[] *arr*) {

        if (*arr* == null || *arr*.length == 0 || *arr*[0] == null) {

            return;

        }

        root = new **Node**(*arr*[0]);

        Queue<Node> queue = new **LinkedList**<>();

        queue.**add**(root);

        int i = 1;

        while (i < *arr*.length) {

            Node current = queue.**poll**();

            if (i < *arr*.length && *arr*[i] != null) {

                current.left = new **Node**(*arr*[i]);

                queue.**add**(current.left);

            }

            i++;

            if (i < *arr*.length && *arr*[i] != null) {

                current.right = new **Node**(*arr*[i]);

                queue.**add**(current.right);

            }

            i++;

        }

    }

    public List<Integer> **rightView**() {

        List<Integer> res = new **ArrayList**<>();

        Queue<Node> q = new **LinkedList**<>();

        q.**add**(root);

        while (!q.**isEmpty**()) {

            int n = q.**size**();

            int num = -1;

            for (int i = 0; i < n; i++) {

                Node curr = q.**poll**();

                num = curr.data;

                if (curr.left != null)

                    q.**add**(curr.left);

                if (curr.right != null)

                    q.**add**(curr.right);

            }

            res.**add**(num);

        }

        return res;

    }

    public List<Integer> **LeftView**() {

        List<Integer> res = new **ArrayList**<>();

        Queue<Node> q = new **LinkedList**<>();

        q.**add**(root);

        while (!q.**isEmpty**()) {

            int n = q.**size**();

            int num = -1;

            for (int i = 0; i < n; i++) {

                Node curr = q.**poll**();

                if (num == -1)

                    num = curr.data;

                if (curr.left != null)

                    q.**add**(curr.left);

                if (curr.right != null)

                    q.**add**(curr.right);

            }

            res.**add**(num);

        }

        return res;

    }

    public List<Integer> **bottomView**() {

        if (root == null) {

            return new **ArrayList**<>();

        }

        Map<Integer, Integer> map = new **TreeMap**<>();

        Queue<Pair> queue = new **LinkedList**<>();

        queue.**add**(new **Pair**(root, 0));

        while (!queue.**isEmpty**()) {

            Pair current = queue.**poll**();

            Node temp = current.node;

            int vertical = current.vertical;

            map.**put**(vertical, temp.data);

            if (temp.left != null) {

                queue.**add**(new **Pair**(temp.left, vertical - 1));

            }

            if (temp.right != null) {

                queue.**add**(new **Pair**(temp.right, vertical + 1));

            }

        }

        List<Integer> result = new **ArrayList**<>();

        for (int value : map.**values**()) {

            result.**add**(value);

        }

        return result;

    }

    public List<Integer> **topView**() {

        if (root == null) {

            return new **ArrayList**<>();

        }

        Map<Integer, Integer> map = new **TreeMap**<>();

        Queue<Pair> queue = new **LinkedList**<>();

        queue.**add**(new **Pair**(root, 0));

        while (!queue.**isEmpty**()) {

            Pair current = queue.**poll**();

            Node temp = current.node;

            int vertical = current.vertical;

            if (!map.**containsKey**(vertical)) {

                map.**put**(vertical, temp.data);

            }

            if (temp.left != null) {

                queue.**add**(new **Pair**(temp.left, vertical - 1));

            }

            if (temp.right != null) {

                queue.**add**(new **Pair**(temp.right, vertical + 1));

            }

        }

        List<Integer> result = new **ArrayList**<>();

        for (int key : map.**keySet**()) {

            result.**add**(map.**get**(key));

        }

        return result;

    }

}

class Pair {

    Node node;

    int vertical;

    public **Pair**(Node *node*, int *vertical*) {

        this.node = *node*;

        this.vertical = *vertical*;

    }

}

public class Main {

    public static void **main**(String[] *args*) {

        Scanner scanner = new **Scanner**(System.in);

        String input = scanner.**nextLine**();

        String[] arr = input.**split**(",");

        Integer[] treeArray = new Integer[arr.length];

        for (int i = 0; i < arr.length; i++) {

            if (arr[i].**trim**().**equals**("null")) {

                treeArray[i] = null;

            } else {

                treeArray[i] = Integer.**parseInt**(arr[i].**trim**());

            }

        }

        BinaryTree tree = new **BinaryTree**();

        tree.**constructBT**(treeArray);

        List<Integer> topView = tree.**topView**();

        System.out.**println**("top view of the tree");

        System.out.**println**(topView);

        System.out.**println**("bottom view of the tree");

        List<Integer> bottomView = tree.**bottomView**();

        System.out.**println**(bottomView);

        System.out.**println**("right view of the tree");

        List<Integer> rightView = tree.**rightView**();

        System.out.**println**(rightView);

        System.out.**println**("Left view of the tree");

        List<Integer> leftView = tree.**LeftView**();

        System.out.**println**(leftView);

        scanner.**close**();

    }

}

OUTPUT:  
