```java
package com.bo.structure;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class BinaryTree {

    class Node{
        int value;
        Node left;
        Node right;
        Node(int v){
            this.value = v;
            left = null;
            right = null;
        }
    }

    private Node root;

    public BinaryTree(int[] arr){
        for(int i:arr){
            insert(i);
        }
    }

    public void insert(int i){
        root = insert(root, i);
    }

    public Node insert(Node node, int i){
        if(node == null){
            node = new Node(i);
        }else if(i <= node.value){
            node.left = insert(node.left, i);
        }else{
            node.right = insert(node.right, i);
        }

        return node;
    }

    public void visit(Node p){
        System.out.println(p.value);
    }

    public void pre(Node p){
        if(p != null){
            visit(p);
            pre(p.left);
            pre(p.right);
        }
    }

    public void ppp(){
        pre(root);
    }

    public void preOrder(Node p){
        Stack<Node> stack = new Stack<Node>();
        if(p != null){
            stack.push(p);
            while(!stack.isEmpty()){
                Node t = stack.pop();
                visit(t);
                if(p.right != null){
                    stack.push(p.right);
                }
                if(p.left != null){
                    stack.push(p.left);
                }
            }
        }
    }

    //first visit then push into stack
    //much more easier to understand
    public void preOrder2(Node p){
        Stack<Node> stack = new Stack<Node>();
        Node node = p;
        while(node != null || !stack.isEmpty()){
            while(node != null){
                visit(node);
                stack.push(node);
                node = node.left;
            }
            if(!stack.isEmpty()){
                node = stack.pop();
                node = node.right;
            }
        }

    }

    public void inOrder(Node p){
        Stack<Node> stack = new Stack<Node>();
        while(p != null){
            while(p != null){
                if(p.right != null)
                    stack.push(p.right);
                stack.push(p);
                p = p.left;
            }
            p = stack.pop();
            while(!stack.isEmpty() && p.right == null){
                visit(p);
                p = stack.pop();
            }
            visit(p);
            if(!stack.isEmpty())
                p = stack.pop();
            else
                p = null;

        }
    }

    public void inOrder2(Node p){
        Stack<Node> stack = new Stack<Node>();
        Node node = p;
        while(node != null || !stack.isEmpty()){
            while(node != null){
                stack.push(node);
                node = node.left;
            }
            if(!stack.isEmpty()){
                node = stack.pop();
                visit(node);
                node = node.right;

            }
        }
    }

    public void IOS(){
        inOrder(root);
    }

    public void postOrder(Node p){
        Node q = p;
        Stack<Node> stack = new Stack<Node>();
        while(p != null){
            //stack in left tree
            for(; p.left != null; p = p.left){
                stack.push(p);
            }
            //current node have no right children or right children have been visit
            while(p != null && (p.right == null || p.right == q)){
                visit(p);
                q = p;
                if(stack.isEmpty())
                    return;
                p = stack.pop();
            }
            stack.push(p);
            p = p.right;
        }
    }

    //double stack
    public void postOrder2(Node p){
        Stack<Node> leftstack = new Stack<Node>();
        Stack<Node> rightstack = new Stack<Node>();
        Node node = p, right;
        do{
            while(node != null){
                right = node.right;
                leftstack.push(node);
                rightstack.push(right);
                node = node.left;
            }
            node = leftstack.pop();
            right = rightstack.pop();
            if(right == null){
                visit(node);
            }else{
                leftstack.push(node);
                rightstack.push(null);
            }
            node = right;

        }while(!leftstack.isEmpty() || !rightstack.isEmpty());
    }

    //single stack
    public void postOrder3(Node p){
        Stack<Node> stack = new Stack<Node>();
        Node node = p, prev = p;
        while (node != null || stack.size() > 0) {
            while (node != null) {
                stack.push(node);
                node = node.left;
            }
            if (stack.size() > 0) {
                Node temp = stack.peek().right;
                if (temp == null || temp == prev) {
                    node = stack.pop();
                    visit(node);
                    prev = node;
                    node = null;
                } else {
                    node = temp;
                }
            }

        }
    }

    //double stack 2
    public void postOrder4(Node p){
        Stack<Node> stack = new Stack<Node>();
        Stack<Node> temp = new Stack<Node>();
```

```java
        Node node = p;
        while (node != null || stack.size() > 0) {
            while (node != null) {
                temp.push(node);
                stack.push(node);
                node = node.right;
            }
            if (stack.size() > 0) {
                node = stack.pop();
                node = node.left;
            }
        }
        while (temp.size() > 0) {//把插入序列都插入到了temp。
            node = temp.pop();
            visit(node);
        }
    }

    public void reverseLeftAndRight(Node p){
        if(p == null)
            return;
        if(null == p.left && null == p.right)
            return;
        Node temp = p.left;
        p.left = p.right;
        p.right = temp;
        reverseLeftAndRight(p.left);
        reverseLeftAndRight(p.right);
    }

    //use Queue
    public void reverseAgain(Node p){
        if(p == null)
            return;
        if(null == p.left && null == p.right)
            return;
        Queue<Node> queue = new LinkedList<Node>();
        queue.offer(p);
        Node temp;
        Node q = p;
        while(!queue.isEmpty()){
            if(null != q.left){
                queue.offer(q.left);
            }
            if(null != q.right){
                queue.offer(q.right);
            }
            temp = q.left;
            q.left = q.right;
            q.right = temp;
            q = queue.poll();
        }

    }

    public void reverse(){
        reverseAgain(root);
    }

    public void POS(){
        postOrder2(root);
    }

     public void morris_inorder(Node root) {
            while(root != null) {
                if(root.left != null) {
                    Node temp = root.left;
                    while(temp.right != null && temp.right != root) {
                        temp = temp.right;
                    }
                    if(temp.right == null) {
                        temp.right = root;
                        root = root.left;
                    } else {
                        System.out.print(root.value + " ");
                        temp.right = null;
                        root = root.right;
                    }
                } else {
                    System.out.print(root.value + " ");
                    root = root.right;
                }
            }

    }

    public static void main(String... args){
        int[] arr = {15,6,23,4,7,71,5,50};
        BinaryTree b = new BinaryTree(arr);
        b.reverse();
        b.ppp();

    }
}
```

```java
package com.bo.offer;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;

public class Tree {

    /**
     * 包含子树
     */
    public static boolean IsSubTree(Node first, Node second){
        boolean result = false;
        if (first != null && second != null) {
            if (first.val == second.val) {
                result = DoesContains(first, second);
            }
            if(!result)
                result = IsSubTree(first.left, second);
            if(!result)
                result = IsSubTree(first.right, second);
        }

        return result;
    }

    public static boolean DoesContains(Node first, Node second){
        if (second == null) {
            return true;
        }
        if (first == null) {
            return false;
        }
        if (first.val != second.val) {
            return false;
        }
        return DoesContains(first.left, second.left) && DoesContains(first.right, second.right);
    }

    /**
     * 镜像
     */
    public static void Mirror(Node root){
        if (root == null) {
            return;
        }
        if (root.left ==null && root.right == null) {
            return;
        }

        Node temp = root.right;
        root.right = root.left;
        root.left = temp;
        if (root.left != null) {
            Mirror(root.left);
        }
        if (root.right != null) {
            Mirror(root.right);
        }
    }

    /**
     * 层次遍历
     */
    public static void LevelPrint(Node root){
        if(root == null)
            return;
        Queue<Node> queue = new LinkedList<>();
        queue.offer(root);
        while(!queue.isEmpty()){
            Node node = queue.poll();
            System.out.print(node.val + " ");
            if (node.left != null) {
                queue.offer(node.left);
            }
            if(node.right != null){
                queue.offer(node.right);
            }
        }
    }

    /**
     * 判断输入序列是不是某二叉树的后续遍历
     */
    public static boolean VerifySequenceOfBST(int[] sequence){
        if (sequence.length < 1) {
            return false;
        }

        int root = sequence[sequence.length-1];
        int i = 0;
        for (;i < sequence.length - 1; i++) {
            if(sequence[i] > root)
                break;
        }
        int j=i;
        for (j = i; j < sequence.length-1; j++) {
            if(sequence[j] < root)
                return false;
        }
        //这里注意使用Arrays.copyOfRange的时候开始下标和结束下标 实际取得元素是 sequence[start....end-1]
        boolean left = true;
        if (i> 0) {
            left = VerifySequenceOfBST(Arrays.copyOfRange(sequence, 0, i));
        }
        boolean right = true;
        if (i < sequence.length -1) {
            right = VerifySequenceOfBST(Arrays.copyOfRange(sequence, i, sequence.length-i));
        }

        return (left && right);
    }

    /**
     * 二叉树中和为某一值得路径
     */
    public static void FindPath(Node node, int expect){
        if (node == null) {
            return;
        }
        LinkedList<Integer> path = new LinkedList<Integer>();
        int currentsum = 0;
        FindPath(node, expect, path, currentsum);
    }

    public static void FindPath(Node root, int expect, LinkedList<Integer> path, int currentsum){
        currentsum += root.val;
        path.add(root.val);

        boolean isleaf = root.left == null && root.right == null;
        if (currentsum == expect && isleaf) {
            //输出list中的路径
            for(int i:path)
                System.out.print(i+" ");
            System.out.println();
        }
        if (root.left != null) {
            FindPath(root.left, expect, path, currentsum);
        }
        if(root.right != null){
            FindPath(root.right, expect, path, currentsum);
        }

        path.removeLast();
    }

    public static void main(String[] args) {
        int[] first = {8,8,7,9,2,-1,-1,-1,-1,4,7};
//      int[] second = {8,9,2};
        Node n_first = LevelConstruct(first);
//      Node n_second = LevelConstruct(second);
//
//      System.out.println(IsSubTree(n_first, n_second));
//      LevelPrint(n_first);
//      int[] sequence = {5,7,6,9,11,10,8};
//      System.out.println(VerifySequenceOfBST(sequence));

        int[] bst = {10,5,12,4,7};
        Node root = LevelConstruct(bst);
        FindPath(root, 22);
    }

    //层次构建子树
    public static Node LevelConstruct(int[] data){
        Node[] nodes = new Node[data.length];;
        if (data.length > 0) {
            for (int i = 0; i < nodes.length; i++) {
                if(data[i] == -1){
                    nodes[i] = null;
                }else
```

```java
                    nodes[i] = new Node(data[i]);
            }
        }
        for (int i= (data.length-2)/2; i>=0;i--) {
            Node node = nodes[i];
            node.left = nodes[2*i+1];
            node.right = nodes[2*i+2];
        }
        return nodes[0];
    }

    //构建二叉搜索树
    public static Node BinarySearchTree(int[] data){
        if (data.length < 1) {
            return null;
        }
        Node root = new Node(data[0]);
        for (int i = 1; i < data.length; i++) {
            Node node = new Node(data[i]);
            Node move = root;
            while(move != null){
                if (move.val < node.val) {
                    move = move.right;
                }else
                    move = move.left;
            }

        }
    }
}

class Node{
    int val;
    Node left;
    Node right;

    public Node(){}

    public Node(int val){
        this.val = val;
        this.left = null;
        this.right = null;
    }
}
```