```
package algorithm.tree;

/**
 * @author xiaobaoqiu  Date: 16-6-29 Time:  下午11:17
 */
public class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}
```

balanced-binary-tree

```
package algorithm.tree;

/**
 * https://leetcode.com/problems/balanced-binary-tree/
 *

 Given a binary tree, determine if it is height-balanced.

 For this problem, a height-balanced binary tree is defined as a binary tre
 in which the depth of the two subtrees of every node
 never differ by more than 1.

 * @author xiaobaoqiu  Date: 16-6-1 Time:  下午10:28
 */
public class BalancedBinaryTree {
    public static void main(String[] args) {

    }

    /**
     * 1 ms
     * Your runtime beats 81.84% of javas ubmissions
     */
    public static boolean isBalanced(TreeNode root) {
        return height(root) != -1;
    }

    /**
     * 树的高度
     * 如果不是平衡树,就返回-1
     */
    private static int height(TreeNode root) {
        if (root == null) return 0;
        int left = height(root.left);
        if (left == -1) return -1;
        int right = height(root.right);
        if (right == -1) return -1;
        if (left + 1 < right || left -1 > right) return -1;
        return Math.max(left, right) + 1;
    }
}
```

binary-tree-inorder-traversal

```
package algorithm.tree;

import java.util.Collection;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.util.Stack;

/**
 * https://leetcode.com/problems/binary-tree-inorder-traversal/
 *
 Given a binary tree, return the inorder traversal of its nodes' values.

 For example:
 Given binary tree {1,#,2,3},
 1
 \
 2
 /
 3
 return [1,3,2].

 Note: Recursive solution is trivial, could you do it iteratively?

 * @author xiaobaoqiu  Date: 16-5-25 Time:  下午9:11
 */
public class BinaryTreeInorderTraversal {
    public static void main(String[] args) {

    }

    /**
     * 递归
     *
     * 1 ms
     * Your runtime beats 62.04% of java submissions
     */
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> path = new LinkedList<Integer>();
        inorder(root, path);
        return path;
    }

    private void inorder(TreeNode node, List<Integer> path) {
        if (node == null) return;

        inorder(node.left, path);
        path.add(node.val);
        inorder(node.right, path);
    }

    /**
     * 非递归，使用栈
```

```
     *
     * 2 ms
     * Your runtime beats 3.35% of java submissions.
     */
    public List<Integer> inorderTraversal_1(TreeNode root) {
        Stack<TreeNode> order = new Stack<TreeNode>();
        List<Integer> path = new LinkedList<Integer>();
        TreeNode node = root;
        while(node != null || !order.isEmpty()) {
            if (node != null) {
                order.push(node);
                node = node.left;
            }
            else {
                node = order.pop();
                path.add(node.val);
                node = node.right;
            }
        }
        return path;
    }
}
```

binary-tree-level-order-traversal

```
package algorithm.tree;

import java.util.ArrayDeque;
import java.util.Collections;
import java.util.Deque;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

/**
 * https://leetcode.com/problems/binary-tree-level-order-traversal/
 * <p/>
 * Given a binary tree, return the level order traversal of its nodes' valu
 * <p/>
 * For example:
 * Given binary tree [3,9,20,null,null,15,7],
 * 3
 * / \
 * 9  20
 * /  \
 * 15   7
 * return its level order traversal as:
 * [
 * [3],
 * [9,20],
 * [15,7]
 * ]
 *
 * @author xiaobaoqiu  Date: 16-6-1 Time:  下午10:58
 */
public class BinaryTreeLevelOrderTraversal {
    public static void main(String[] args) {
        //[3,9,20,null,null,15,7]
        TreeNode node3 = new TreeNode(3);
        TreeNode node9 = new TreeNode(9);
        TreeNode node20 = new TreeNode(20);
        TreeNode node15 = new TreeNode(15);
        TreeNode node7 = new TreeNode(7);
        node3.left = node9;
        node3.right = node20;
        node20.left = node15;
        node20.right = node7;

        List<List<Integer>> ret = levelOrder(node3);
        for(List<Integer> item : ret) {
            System.out.println(item);
        }
    }

    /**
     * 3 ms
     * Your runtime beats 13.20% of java submissions
     */
    public static List<List<Integer>> levelOrder(TreeNode root) {
```

```
        if (root == null) return Collections.emptyList();

        List<List<Integer>> ret = new LinkedList<List<Integer>>();
        Queue<TreeNode> queue = new ArrayDeque<TreeNode>();
        queue.add(root);
        int next = 0, current = 1;
        List<Integer> cur = new LinkedList<Integer>();
        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            current--;
            cur.add(node.val);

            if (node.left != null) {queue.add(node.left);next++;}
            if (node.right != null) {queue.add(node.right);next++;}
            if (current == 0) {
                ret.add(cur);
                cur = new LinkedList<Integer>();
                current = next;
                next = 0;
            }
        }
        return ret;
    }
}
```

binary-tree-level-order-traversal-ii

```
package algorithm.tree;

import java.util.ArrayDeque;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

/**
 * https://leetcode.com/problems/binary-tree-level-order-traversal-ii/
 *
 Given a binary tree, return the bottom-up level order traversal of its noc
 (ie, from left to right, level by level from leaf to root).

 For example:
 Given binary tree {3,9,20,#,#,15,7},
 3
 / \
 9  20
 /  \
 15   7
 return its bottom-up level order traversal as:
 [
 [15,7],
 [9,20],
 [3]
 ]

 * @author xiaobaoqiu  Date: 16-6-1 Time:  下午10:58
 */
public class BinaryTreeLevelOrderTraversalII {
    public static void main(String[] args) {
        //[3,9,20,null,null,15,7]
        TreeNode node3 = new TreeNode(3);
        TreeNode node9 = new TreeNode(9);
        TreeNode node20 = new TreeNode(20);
        TreeNode node15 = new TreeNode(15);
        TreeNode node7 = new TreeNode(7);
        node3.left = node9;
        node3.right = node20;
        node20.left = node15;
        node20.right = node7;

        List<List<Integer>> ret = levelOrderBottom(node3);
        for(List<Integer> item : ret) {
            System.out.println(item);
        }
    }

    /**
     * 3 ms
     * Your runtime beats 31.02% of java submissions
     */
    public static List<List<Integer>> levelOrderBottom(TreeNode root) {
```

```
        if (root == null) return Collections.emptyList();

        List<List<Integer>> ret = new LinkedList<List<Integer>>();
        Queue<TreeNode> queue = new ArrayDeque<TreeNode>();
        queue.add(root);
        int next = 0, current = 1;
        List<Integer> cur = new LinkedList<Integer>();
        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            current--;
            cur.add(node.val);

            if (node.left != null) {queue.add(node.left);next++;}
            if (node.right != null) {queue.add(node.right);next++;}
            if (current == 0) {
                ret.add(0, cur);
                cur = new LinkedList<Integer>();
                current = next;
                next = 0;
            }
        }
        return ret;
    }
}
```

binary-tree-paths

```
package algorithm.tree;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * https://leetcode.com/problems/binary-tree-paths/
 *
 Given a binary tree, return all root-to-leaf paths.

 For example, given the following binary tree:

 1
 /   \
 2     3
 \
 5
 All root-to-leaf paths are:

 ["1->2->5", "1->3"]

 * @author xiaobaoqiu  Date: 16-7-7 Time:  下午11:13
 */
public class BinaryTreePaths {
    public static void main(String[] args) {
        //[3,9,20,null,null,15,7]
        TreeNode node3 = new TreeNode(3);
        TreeNode node9 = new TreeNode(9);
        TreeNode node20 = new TreeNode(20);
        TreeNode node15 = new TreeNode(15);
        TreeNode node7 = new TreeNode(7);
        node3.left = node9;
        node3.right = node20;
        node20.left = node15;
        node20.right = node7;

        List<String> path = binaryTreePaths(node3);
        System.out.println(path);
    }

    /**
     * 3 ms
     * Your runtime beats 28.89% of java submissions
     */
    public static List<String> binaryTreePaths(TreeNode root) {
        List<String> pathList = new ArrayList<String>();
        if (root != null) dfs(root, pathList, null);
        return pathList;
    }

    private static void dfs(TreeNode node, List<String> pathList, String cu
        if (cur == null) cur = "";
        else cur += "->";
```

```
        cur += node.val;

        if (node.left == null && node.right == null) pathList.add(cur);
        else {
            if (node.left != null) dfs(node.left, pathList, cur);
            if (node.right != null) dfs(node.right, pathList, cur);
        }
    }
}
```

binary-tree-preorder-traversal

```
package algorithm.tree;

import java.util.LinkedList;
import java.util.List;
import java.util.Stack;

/**
 *
 https://leetcode.com/problems/binary-tree-preorder-traversal/

 Given a binary tree, return the preorder traversal of its nodes' values.

 For example:
 Given binary tree {1,#,2,3},
 1
 \
 2
 /
 3
 return [1,2,3].

 Note: Recursive solution is trivial, could you do it iteratively?

 * @author xiaobaoqiu  Date: 16-5-24 Time:  下午8:50
 */
public class BinaryTreePreorderTraversal {
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        TreeNode left = new TreeNode(2);
        TreeNode leftleft = new TreeNode(3);
        left.left = leftleft;
        root.left = left;

//        System.out.println(preorderTraversal(root));
        System.out.println(preorderTraversal_1(root));
    }

    /**
     * 递归
     * root --> left child --> right child
     *
     * 1 ms
     * Your runtime beats 56.04% of java submissions
     */
    public static List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> path = new LinkedList<Integer>();
        preorder(root, path);
        return path;
    }

    private static void preorder(TreeNode root, List<Integer> path) {
        if (root == null) return;
        path.add(root.val);
        preorder(root.left, path);
```

```
        preorder(root.right, path);
    }

    /**
     * 非递归，使用栈
     * [2,3,null,1]
     *
     * 2 ms
     * Your runtime beats 1.25% of java submissions
     */
    public static List<Integer> preorderTraversal_1(TreeNode root) {
        Stack<TreeNode> order = new Stack<TreeNode>();
        List<Integer> path = new LinkedList<Integer>();
        if (root != null) order.push(root);
        while(!order.empty()) {
            TreeNode node = order.pop();
            path.add(node.val);
            if (node.right != null) order.add(node.right);
            if (node.left != null) order.add(node.left);
        }
        return path;
    }
}
```

invert-binary-tree

```
package algorithm.tree;

/**
 * https://leetcode.com/problems/invert-binary-tree/
 *
 Invert a binary tree.

 4
 /   \
 2     7
 / \   / \
 1   3 6   9

 to

 4
 /   \
 7     2
 / \   / \
 9   6 3   1

 Trivia:
 This problem was inspired by this original tweet by Max Howell:
 Google: 90% of our engineers use the software you wrote (Homebrew), but yo

 * @author xiaobaoqiu  Date: 16-5-19 Time:  下午9:46
 */
public class InvertBinaryTree {
    public static void main(String[] args) {

    }

    /**
     * 1 ms
     * Your runtime beats 0.52% of java submissions
     */
//    public TreeNode invertTree(TreeNode root) {
//        if (root == null) return root;
//        if (root.left == null && root.right == null) return root;
//
//        TreeNode newLeft = invertTree(root.right);
//        TreeNode newRight = invertTree(root.left);
//        root.left = newLeft;
//        root.right = newRight;
//        return root;
//    }


    /**
     * 0 ms
     * Your runtime beats 21.92% of java submissions
     */
    public TreeNode invertTree(TreeNode root) {
        if (root == null) return root;
```

```
        if(root.left != null || root.right != null) {
            TreeNode newRight = invertTree(root.left);
            root.left = invertTree(root.right);
            root.right = newRight;
        }
        return root;
    }
}
```

lowest-common-ancestor-of-a-binary-search-tree

```
package algorithm.tree;

/**
 * https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-
 *
 Given a binary search tree (BST), find the lowest common ancestor (LCA) of

 According to the definition of LCA on Wikipedia:
 “The lowest common ancestor is defined between two nodes v and w as the lo
 (where we allow a node to be a descendant of itself).”

 _______6______
 /              \
 ___2__          ___8__
 /      \        /      \
 0      _4       7       9
 /  \
 3   5
 For example, the lowest common ancestor (LCA) of nodes 2 and 8 is 6.
 Another example is LCA of nodes 2 and 4 is 2, since a node can be a descen
 * @author xiaobaoqiu  Date: 16-5-25 Time:  下午11:09
 */
public class LowestCommonAncestorOfABinarySearchTree {
    public static void main(String[] args) {
        //[5,3,6,2,4,null,null,1]
        TreeNode node5 = new TreeNode(5);
        TreeNode node3 = new TreeNode(3);
        TreeNode node6 = new TreeNode(6);
        TreeNode node2 = new TreeNode(2);
        TreeNode node4 = new TreeNode(4);
        TreeNode node1 = new TreeNode(1);
        node5.left = node3;
        node5.right = node6;
        node3.left = node2;
        node3.right = node4;
        node2.left = node1;

        TreeNode ret = lowestCommonAncestor(node5, node1, node4);
        System.out.println(ret.val);
    }

    /**
     * 10 ms
     * Your runtime beats 47.57% of java submissions
     */
    public static TreeNode lowestCommonAncestor(TreeNode root, TreeNode p,
        if (root.val < p.val && root.val < q.val) return lowestCommonAncest
        if (root.val > p.val && root.val > q.val) return lowestCommonAncest
        return root;
    }
}
```

maximum-depth-of-binary-tree

```
package algorithm.tree;

/**
 * https://leetcode.com/problems/maximum-depth-of-binary-tree/

 Given a binary tree, find its maximum depth.

 The maximum depth is the number of nodes along the longest path from the r

 Subscribe to see which companies asked this question

 * @author xiaobaoqiu  Date: 16-5-17 Time:  下午9:40
 */
public class MaximumDepthOfBinaryTree {
    public static void main(String[] args) {

    }

    /**
     * 1 ms
     * Your runtime beats 10.20% of java submissions
     */
    public int maxDepth(TreeNode root) {
        if (root == null) return 0;
        if (root.left == null && root.right == null) return 1;
        return Math.max(maxDepth(root.left), maxDepth(root.right)) + 1;
    }
}
```

minimum-depth-of-binary-tree

```
package algorithm.tree;

/**
 * https://leetcode.com/problems/minimum-depth-of-binary-tree/
 *
 Given a binary tree, find its minimum depth.

 The minimum depth is the number of nodes along the shortest path from the

 Subscribe to see which companies asked this question

 * @author xiaobaoqiu  Date: 16-7-1 Time:  上午12:35
 */
public class MinimumDepthOfBinaryTree {
    public static void main(String[] args) {
        //[3,9,20,null,null,15,7]
        TreeNode node3 = new TreeNode(3);
        TreeNode node9 = new TreeNode(9);
        TreeNode node20 = new TreeNode(20);
        TreeNode node15 = new TreeNode(15);
        TreeNode node7 = new TreeNode(7);
        node3.left = node9;
        node3.right = node20;
        node20.left = node15;
        node20.right = node7;

        System.out.println(minDepth(node3));
    }

    /**
     * DFS +  剪枝
     *
     * 1 ms
     * Your runtime beats 10.62% of java submissions
     */
    public static int minDepth(TreeNode root) {
        if (root == null) return 0;
        return dfs(root, 0, Integer.MAX_VALUE);
    }

    private static int dfs(TreeNode node, int curDepth, int curMin) {
        curDepth += 1;

        //fail fast
        if (curDepth >= curMin) return curMin;

        //leaf node
        if (node.left == null && node.right == null) {
            if (curDepth < curMin) return curDepth;
        }

        if (node.left != null) {
            int leftDepth = dfs(node.left, curDepth, curMin);
            if (leftDepth < curMin) curMin = leftDepth;
```

```
        }
        if (node.right != null) {
            int rightDepth = dfs(node.right, curDepth, curMin);
            if (rightDepth < curMin) curMin = rightDepth;
        }
        return curMin;
    }
}
```

path-sum

```
package algorithm.tree;

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

/**
 * https://leetcode.com/problems/path-sum/
 *
 Given a binary tree and a sum, determine if the tree has a root-to-leaf pa
 such that adding up all the values along the path equals the given sum.

 For example:
 Given the below binary tree and sum = 22,
 5
 / \
 4   8
 /   / \
 11  13  4
 /  \      \
 7    2      1
 return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 2

 * @author xiaobaoqiu  Date: 16-6-29 Time:  下午11:16
 */
public class PathSum {
    public static void main(String[] args) {
        //[1,-2,-3,1,3,-2,null,-1]  --> 2
        TreeNode node1 = new TreeNode(1);
        TreeNode node_2 = new TreeNode(-2);
        TreeNode node_3 = new TreeNode(-3);
        TreeNode node1_1 = new TreeNode(1);
        TreeNode node3 = new TreeNode(3);
        TreeNode node_2_1 = new TreeNode(-2);
        TreeNode node_1 = new TreeNode(-1);
        node1.left = node_2;
        node1.right = node_3;
        node_2.left = node1_1;
        node_2.right = node3;
        node_3.left = node_2_1;
        node1_1.left = node_1;

        System.out.println(hasPathSum(node1, 2));
    }

    /**
     * DFS
     * 1 ms
     * Your runtime beats 10.21% of java submissions
     */
    public static boolean hasPathSum(TreeNode root, int sum) {
        if (root == null) return false;
//        String curPath = "";
//        return dfs_debug(root, sum, 0, curPath);
```

```
        return dfs(root, sum, 0);
    }

    public static boolean dfs(TreeNode root, int expect, int curSum) {
        curSum += root.val;

        // leaf node
        if (root.left == null && root.right == null) {
            if (curSum == expect) return true;
        }
        if (root.left != null && dfs(root.left, expect, curSum)) return tru
        return root.right != null && dfs(root.right, expect, curSum);
    }

    public static boolean dfs_debug(TreeNode root, int expect, int curSum,
        curSum += root.val;

        // leaf node
        if (root.left == null && root.right == null) {
            System.out.println(curPath + " --> " + root.val + " = " + (curS
            if (curSum == expect) return true;
        }
        if (root.left != null && dfs_debug(root.left, expect, curSum, curPa
        return root.right != null && dfs_debug(root.right, expect, curSum,
    }
}
```

same-tree

```
package algorithm.tree;

/**
 * https://leetcode.com/problems/same-tree/
 *
 Given two binary trees, write a function to check if they are equal or not

 Two binary trees are considered equal if they are structurally identical a

 * @author xiaobaoqiu  Date: 16-5-18 Time:  下午9:19
 */
public class SameTree {
    public static void main(String[] args) {

    }

    /**
     * 0 ms
     * Your runtime beats 14.90% of java submissions.
     */
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if ( p == null && q == null) return true;
        if (p == null || q == null) return false;

        //到这里说明 p != null && q != null
        if (p.val != q.val) return false;

        return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
    }
}
```

symmetric-tree

```
package algorithm.tree;

/**
 * https://leetcode.com/problems/symmetric-tree/
 *
 Given a binary tree, check whether it is a mirror of itself
 (ie, symmetric around its center).

 For example, this binary tree is symmetric:

 1
 / \
 2   2
 / \ / \
 3  4 4  3
 But the following is not:
 1
 / \
 2   2
 \   \
 3    3
 Note:
 Bonus points if you could solve it both recursively and iteratively.
 * @author xiaobaoqiu  Date: 16-6-1 Time:  下午10:42
 */
public class SymmetricTree {
    public static void main(String[] args) {

    }

    /**
     * 1 ms
     * Your runtime beats 23.80% of java submissions
     */
    public static boolean isSymmetric(TreeNode root) {
        if (root == null) return true;
        return isSymmetric(root.left, root.right);
    }

    private static boolean isSymmetric(TreeNode left, TreeNode right) {
        if (left == null && right == null) return true;
        if (left == null || right == null) return false;
        return left.val == right.val && isSymmetric(left.left, right.right)
    }
}
```