

# 目录

第 1 章	Tree	1	1.6	Lowest Common Ancestor of a Binary Search Tree	6
1.1	Maximum Depth of Binary Tree	1	1.7	Balanced Binary Tree	7
1.2	Invert Binary Tree	2	1.8	Binary Tree Maximum Path Sum	8
1.3	Same Tree	3	1.9	Populating Next Right Pointers in Each Node	9
1.4	Binary Search Tree Iterator	4	1.10	Convert Sorted Array to Binary Search Tree	11
1.5	Unique Binary Search Trees	5			



# 第 1 章

## Tree

### 1.1 Maximum Depth of Binary Tree

#### Description

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

#### Solution

```
private int maxDepth(TreeNode node) {  
    if (node == null) {  
        return 0;  
    }  
    return Math.max(maxDepth(node.left), maxDepth(node.right)) + 1;  
}
```

## 1.2 Invert Binary Tree

### Description

Invert a binary tree.



### Solution I

```

public TreeNode invertTree(TreeNode root) {
    if (root == null) {
        return null;
    }

    TreeNode right = root.right;
    root.right = invertTree(root.left);
    root.left = invertTree(right);
    return root;
}

```

### Solution II

```

public TreeNode invertTree(TreeNode root) {
    if (root == null) {
        return null;
    }

    Queue<TreeNode> queue = new LinkedList<TreeNode>();
    queue.offer(root);

    while (!queue.isEmpty()) {
        TreeNode node = queue.poll();

        TreeNode left = node.left;
        node.left = node.right;
        node.right = left;

        if (node.left != null) {
            queue.offer(node.left);
        }

        if (node.right != null) {
            queue.offer(node.right);
        }
    }

    return root;
}

```

## 1.3 Same Tree

### Description

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 and the nodes have the same value.

### Solution

```
public boolean isSameTree(TreeNode p, TreeNode q) {  
    if (p == null && q == null) {  
        return true;  
    }  
    if (p == null || q == null) {  
        return false;  
    }  
    return p.val == q.val && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);  
}
```

## 1.4 Binary Search Tree Iterator

### Description

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST. Calling next() will return the next smallest number in the BST.

**Note:** next() and hasNext() should run in average  $O(1)$  time and uses  $O(h)$  memory, where  $h$  is the height of the tree.

### Solution

```
public class BSTIterator {

    private Stack<TreeNode> mStack;
    private TreeNode mCurNode;

    public BSTIterator(TreeNode root) {
        mStack = new Stack<TreeNode>();
        mCurNode = root;
    }

    public boolean hasNext() {
        return !mStack.isEmpty() || mCurNode != null;
    }

    public int next() {
        int result = -1;

        while (hasNext()) {
            if (mCurNode != null) {
                mStack.push(mCurNode);
                mCurNode = mCurNode.left;
            } else {
                mCurNode = mStack.pop();
                result = mCurNode.val;
                mCurNode = mCurNode.right;
                break;
            }
        }

        return result;
    }
}
```

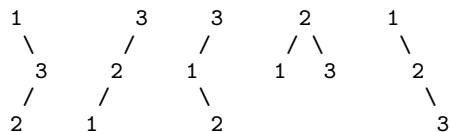
## 1.5 Unique Binary Search Trees

### Description

Given  $n$ , how many structurally unique BST's (binary search trees) that store values  $1 \dots n$ ?

For example,

Given  $n = 3$ , there are a total of 5 unique BST's.



### Solution

```

public int numTrees(int n) {
    int[] dp = new int[n + 1];
    dp[0] = 1;

    for (int i = 1; i <= n; i++) {
        for (int j = 0; j < i; j++) {
            dp[i] += dp[j] * dp[i - j - 1];
        }
    }

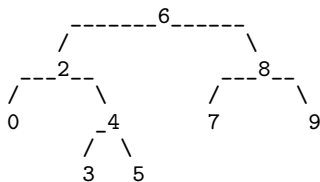
    return dp[n];
}
  
```

## 1.6 Lowest Common Ancestor of a Binary Search Tree

### Description

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

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



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 descendant of itself according to the LCA definition.

### Solution

```

// 耗时 9ms
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if (!checkExist(root, p) || !checkExist(root, q)) {
        throw new IllegalArgumentException("Not exist!!");
    }

    while (root != null) {
        if (p.val < root.val && q.val < root.val) {
            root = root.left;
        } else if (p.val > root.val && q.val > root.val) {
            root = root.right;
        } else {
            break;
        }
    }
    return root;
}

/**
 * 如何判断 p 或 q 一定存在，如果是 BST 就很简单
 */
private boolean checkExist(TreeNode root, TreeNode node) {
    TreeNode cur = root;
    while (cur != null) {
        if (node.val > cur.val) {
            cur = cur.right;
        } else if (node.val < cur.val) {
            cur = cur.left;
        } else {
            return true;
        }
    }
    return false;
}

```



## 1.7 Balanced Binary Tree

### Description

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

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

### Solution

```
public boolean isBalanced(TreeNode root) {
    return isBalanced(root, null);
}

private boolean isBalanced(TreeNode root, int[] height) {
    if (root == null) {
        return true;
    }

    int[] left = new int[1], right = new int[1];

    boolean result = isBalanced(root.left, left) && isBalanced(root.right, right);

    if (height != null) {
        height[0] = Math.max(left[0], right[0]) + 1;
    }

    return result && Math.abs(left[0] - right[0]) <= 1;
}
```

## 1.8 Binary Tree Maximum Path Sum

### Description

Given a binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain at least one node and does not need to go through the root.

For example:

Given the below binary tree,



Return 6.

### Solution

```

public int maxPathSum(TreeNode root) {
    return maxPathSum(root, null);
}

/**
 * max 表示包含 root 的单边路径最大和
 */
private int maxPathSum(TreeNode root, int[] max) {
    if (root == null) {
        return Integer.MIN_VALUE; // 此处容易错
    }
    int[] left = new int[1], right = new int[1];
    int leftMax = maxPathSum(root.left, left);
    int rightMax = maxPathSum(root.right, right);
    if (max != null) {
        max[0] = max(left[0], right[0], 0) + root.val; // 此处容易错
    }

    // 容易错, 要考虑到所有可能的情况
    return max(leftMax, rightMax, root.val, left[0] + right[0] + root.val,
        left[0] + root.val, right[0] + root.val);
}

private int max(int... vals) {
    int max = Integer.MIN_VALUE;
    for (int val : vals) {
        max = Math.max(max, val);
    }
    return max;
}
  
```

## 1.9 Populating Next Right Pointers in Each Node

### Description

Given a binary tree

```
struct TreeLinkNode {
    TreeLinkNode *left;
    TreeLinkNode *right;
    TreeLinkNode *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

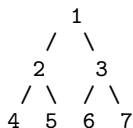
#### Note:

You may only use constant extra space.

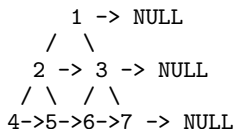
You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

For example,

Given the following perfect binary tree,



After calling your function, the tree should look like:



## Solution I

```

/** 递归法，巧妙地运用 dummy 使代码很简洁
 * 假定当前 root 所在层已连好，要连下一层
 */
public void connect(TreeLinkNode root) {
    if (root == null) {
        return;
    }
    TreeLinkNode dummy = new TreeLinkNode(0), cur = dummy;
    for (TreeLinkNode p = root; p != null; p = p.next) {
        if (p.left != null) {
            cur.next = p.left;
            cur = cur.next;
        }
        if (p.right != null) {
            cur.next = p.right;
            cur = cur.next;
        }
    }
    connect(dummy.next);
}

```

## Solution II

```

/**
 * 将递归转成非递归很简单，就加一层循环，且结尾处加 root = dummy.next 即可
 */
public void connect2(TreeLinkNode root) {
    while (root != null) {
        TreeLinkNode dummy = new TreeLinkNode(0), cur = dummy;

        for (TreeLinkNode p = root; p != null; p = p.next) {
            if (p.left != null) {
                cur.next = p.left;
                cur = cur.next;
            }
            if (p.right != null) {
                cur.next = p.right;
                cur = cur.next;
            }
        }

        root = dummy.next;
    }
}

```

## 1.10 Convert Sorted Array to Binary Search Tree

### Description

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

### Solution

```
public TreeNode sortedArrayToBST(int[] nums) {
    return sortedArrayToBST(nums, 0, nums.length);
}

private TreeNode sortedArrayToBST(int[] nums, int start, int end) {
    if (start >= end) {
        return null;
    }
    int mid = start + (end - start) / 2;
    TreeNode root = new TreeNode(nums[mid]);
    root.left = sortedArrayToBST(nums, start, mid);
    root.right = sortedArrayToBST(nums, mid + 1, end);
    return root;
}
```