Binary Tree Problems

Lowest Common Ancestor of a Binary Tree
Given a binary tree, find the lowest common ancestor (LCA) of two nodes.

Binary Tree Serialization and Deserialization
Implement functions to serialize (convert to a string) and deserialize (reconstruct from a string) a binary tree.

Convert Binary Tree to Doubly Linked List
Convert a binary tree into a doubly linked list in-place.

Find the Maximum Width of a Binary Tree
Determine the maximum width (number of nodes at the widest level) of a binary tree.

Check if Binary Tree is Balanced
Given a binary tree, check if it is height-balanced (the difference in heights of left and right subtrees is no more than 1).

Vertical Order Traversal
Print the vertical order traversal of a binary tree (nodes at the same horizontal distance from the root).

Sum of Left Leaves
Calculate the sum of all leaves that are left children in the binary tree.

Boundary Traversal of Binary Tree
Print the boundary nodes of a binary tree in anti-clockwise order starting from the root.

Find Diameter of Binary Tree
Find the diameter of a binary tree (the length of the longest path between any two nodes).

Kth Smallest Element in a BST
Find the kth smallest element in a binary search tree (BST).

explain problem with example that contains input and output and give java code in optimized and simple form with time complexity

Memory updated

Here's an explanation of each problem with an example, input/output, and the corresponding optimized and simple Java code:

---

1. Lowest Common Ancestor of a Binary Tree

Problem: Given a binary tree, find the lowest common ancestor (LCA) of two nodes p and q. The LCA is the lowest node that has both p and q as descendants.
Example:

```makefile
Input:
       3
      / \
     5   1
    / \ / \
   6  2 0  8
     / \
    7   4

p = 5, q = 1

Output: 3
```

Java Code:

```java
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int x) { val = x; }
}

public class BinaryTreeLCA {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if (root == null || root == p || root == q) return root;
        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);
        if (left != null && right != null) return root;
        return left != null ? left : right;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

2. Binary Tree Serialization and Deserialization

Problem: Implement functions to serialize a binary tree to a string and deserialize it back.
Example:

```csharp
Input:        1
             / \
            2   3
               / \
              4   5

Serialized Output: 1,2,null,null,3,4,null,null,5,null,null
```

Java Code:

```java
public class Codec {
    // Serialize a binary tree
    public String serialize(TreeNode root) {
        if (root == null) return "null,";
        return root.val + "," + serialize(root.left) + serialize(root.right);
    }

    // Deserialize the serialized string back to a tree
    public TreeNode deserialize(String data) {
        Queue<String> nodes = new LinkedList<>(Arrays.asList(data.split(",")));
        return buildTree(nodes);
    }

    private TreeNode buildTree(Queue<String> nodes) {
        String val = nodes.poll();
        if (val.equals("null")) return null;
        TreeNode node = new TreeNode(Integer.parseInt(val));
        node.left = buildTree(nodes);
        node.right = buildTree(nodes);
        return node;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

3. Convert Binary Tree to Doubly Linked List

Problem: Convert a binary tree into a doubly linked list in-place.
Example:

```rust
Input:
        10
       /  \
      12   15
     / \   /
    25 30 36

Output: 25 <-> 12 <-> 30 <-> 10 <-> 36 <-> 15
```

Java Code:

```java
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int x) { val = x; }
}

public class BinaryTreeToDLL {
    TreeNode head = null, prev = null;

    public TreeNode convertToDLL(TreeNode root) {
        if (root == null) return null;
        convertToDLL(root.left);
        if (prev == null) head = root;
        else {
            root.left = prev;
            prev.right = root;
        }
        prev = root;
        convertToDLL(root.right);
        return head;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

4. Find the Maximum Width of a Binary Tree

Problem: Determine the maximum width of a binary tree.
Example:

```vbnet
Input:
       1
      / \
     3   2
    / \   \
   5   3   9

Output: 4 (maximum width is at level 2)
```

Java Code:

```java
public class MaxWidthBinaryTree {
    public int maxWidth(TreeNode root) {
        if (root == null) return 0;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        int maxWidth = 0;

        while (!queue.isEmpty()) {
            int size = queue.size();
            maxWidth = Math.max(maxWidth, size);
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }
        }
        return maxWidth;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

5. Check if Binary Tree is Balanced

Problem: Check if a binary tree is height-balanced.
Example:

```vbnet
Input:
        1
       / \
      2   3
     / \
    4   5

Output: true
```

Java Code:

```java
public class BalancedBinaryTree {
    public boolean isBalanced(TreeNode root) {
        return checkHeight(root) != -1;
    }

    private int checkHeight(TreeNode root) {
        if (root == null) return 0;
        int left = checkHeight(root.left);
        if (left == -1) return -1;
        int right = checkHeight(root.right);
        if (right == -1) return -1;
        if (Math.abs(left - right) > 1) return -1;
        return Math.max(left, right) + 1;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

These are just a few problems with their explanations, input/output, and optimized Java solutions. Let me know if you'd like more problems or detailed code for the remaining ones!

give remaining

Let's continue with the remaining problems for binary trees:

---

6. Vertical Order Traversal

Problem: Print the vertical order traversal of a binary tree (nodes at the same horizontal distance from the root).

Example:

lua

```lua
Input:
       1
      / \
     2   3
    / \   \
   4   5   6

Output: [[4], [2], [1, 5], [3], [6]]
```

Java Code:

java

```java
import java.util.*;

public class VerticalOrderTraversal {
    public List<List<Integer>> verticalOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;

        Map<Integer, List<Integer>> columnMap = new TreeMap<>();
        Queue<Pair> queue = new LinkedList<>();
        queue.add(new Pair(root, 0));

        while (!queue.isEmpty()) {
            Pair pair = queue.poll();
            TreeNode node = pair.node;
            int column = pair.column;

            columnMap.putIfAbsent(column, new ArrayList<>());
            columnMap.get(column).add(node.val);

            if (node.left != null) queue.add(new Pair(node.left, column - 1));
            if (node.right != null) queue.add(new Pair(node.right, column + 1));
        }

        for (List<Integer> col : columnMap.values()) {
            result.add(col);
        }
        return result;
    }

    class Pair {
        TreeNode node;
        int column;
        Pair(TreeNode node, int column) {
            this.node = node;
            this.column = column;
        }
    }
}
```

Time Complexity: O(n log n)
Space Complexity: O(n)

---

7. Sum of Left Leaves

Problem: Calculate the sum of all leaves that are left children in the binary tree.

Example:

makefile

```makefile
Input:
       3
      / \
     9  20
        /  \
       15   7

Output: 24 (9 + 15)
```

Java Code:

java

```java
public class SumOfLeftLeaves {
    public int sumOfLeftLeaves(TreeNode root) {
        if (root == null) return 0;
        int sum = 0;
        if (root.left != null && root.left.left == null && root.left.right == null) {
            sum += root.left.val;
        }
        return sum + sumOfLeftLeaves(root.left) + sumOfLeftLeaves(root.right);
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

8. Boundary Traversal of Binary Tree

Problem: Print the boundary nodes of a binary tree in anti-clockwise order starting from the root.
Example:

makefile

```makefile
Input:
          20
        /   \
       8     22
      / \      \
     4   12     25
        /  \
       10   14

Output: [20, 8, 4, 10, 14, 25, 22]
```

Java Code:

java

```java
public class BoundaryTraversal {
    public List<Integer> boundaryOfBinaryTree(TreeNode root) {
        List<Integer> boundary = new ArrayList<>();
        if (root == null) return boundary;
        if (!isLeaf(root)) boundary.add(root.val);
        addLeftBoundary(root.left, boundary);
        addLeaves(root, boundary);
        addRightBoundary(root.right, boundary);
        return boundary;
    }

    private void addLeftBoundary(TreeNode node, List<Integer> boundary) {
        while (node != null) {
            if (!isLeaf(node)) boundary.add(node.val);
            if (node.left != null) node = node.left;
            else node = node.right;
        }
    }

    private void addLeaves(TreeNode node, List<Integer> boundary) {
        if (isLeaf(node)) {
            boundary.add(node.val);
            return;
        }
        if (node.left != null) addLeaves(node.left, boundary);
        if (node.right != null) addLeaves(node.right, boundary);
    }

    private void addRightBoundary(TreeNode node, List<Integer> boundary) {
        Stack<Integer> stack = new Stack<>();
        while (node != null) {
            if (!isLeaf(node)) stack.push(node.val);
            if (node.right != null) node = node.right;
            else node = node.left;
        }
        while (!stack.isEmpty()) boundary.add(stack.pop());
    }

    private boolean isLeaf(TreeNode node) {
        return node.left == null && node.right == null;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

9. Find Diameter of Binary Tree

Problem: Find the diameter of a binary tree (the length of the longest path between any two nodes).
Example:

```makefile
Input:
       1
      / \
     2   3
    / \
   4   5

Output: 3
```

Java Code:

```java
public class DiameterOfBinaryTree {
    private int diameter = 0;

    public int diameterOfBinaryTree(TreeNode root) {
        depth(root);
        return diameter;
    }

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

Time Complexity: O(n)
Space Complexity: O(n)

---

10. Kth Smallest Element in a BST

Problem: Find the kth smallest element in a binary search tree (BST).
Example:

```makefile
Input:
       5
      / \
     3   6
    / \
   2   4
  /
 1

k = 3

Output: 3
```

Java Code:

```java
public class KthSmallestElement {
    public int kthSmallest(TreeNode root, int k) {
        Stack<TreeNode> stack = new Stack<>();
        while (true) {
            while (root != null) {
                stack.push(root);
                root = root.left;
            }
            root = stack.pop();
            if (--k == 0) return root.val;
            root = root.right;
        }
        }
    }
```

Time Complexity: O(k)
Space Complexity: O(k)

These solutions cover all the remaining problems you requested. Each solution is optimized and includes an example with input and output. Let me know if you need further explanations or modifications!

Convert BST to Greater Tree
Convert a binary search tree to a greater tree where each node's value is replaced by the sum of all greater values.

Binary Tree Tilt
Compute the tilt of a binary tree where the tilt of a node is the absolute difference between the sums of its left and right subtrees.

Find All Nodes Distance K from Target
Given a target node, find all nodes that are at distance K from the target node in a binary tree.

Construct Binary Tree from Inorder and Postorder Traversal
Construct a binary tree given its inorder and postorder traversals.

Check if Binary Tree is a Subtree of Another Binary Tree
Determine if one binary tree is a subtree of another binary tree.

Binary Tree Paths
Find all root-to-leaf paths in a binary tree.

Mirror of Binary Tree
Find the mirror image of a binary tree (i.e., flip the tree horizontally).

Max Path Sum in Binary Tree
Find the maximum path sum (the sum of the maximum sum path between any two nodes) in a binary tree.

Find the Closest Binary Search Tree Value
Given a binary search tree and a target value, find the value in the tree that is closest to the target.

Flatten Binary Tree to Linked List
Flatten a binary tree to a linked list in-place using the right child pointer as the next pointer.

Memory updated

Here’s an explanation of each remaining problem with an example, input/output, and the corresponding optimized and simple Java code:

---

1. Convert BST to Greater Tree

Problem: Convert a binary search tree (BST) to a greater tree where each node's value is replaced by the sum of all greater values in the tree.
Example:

```makefile
Input:
       4
      / \
     1   6
    / \ / \
   0  2 5  7
            \
             8

Output:
       30
      / \
    36  21
    / \ / \
   36 35 26 15
            \
             8
```

Java Code:

```java
public class ConvertBSTToGreaterTree {
    private int sum = 0;

    public TreeNode convertBST(TreeNode root) {
        if (root != null) {
            convertBST(root.right);
            sum += root.val;
            root.val = sum;
            convertBST(root.left);
        }
        return root;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

2. Binary Tree Tilt

Problem: Compute the tilt of a binary tree. The tilt of a tree node is the absolute difference between the sum of all nodes in its left subtree and right subtree.
Example:

```makefile
Input:
       1
      / \
     2   3

Output: 1 (|2 - 3| = 1)
```

Java Code:

```java
public class BinaryTreeTilt {
    private int totalTilt = 0;

    public int findTilt(TreeNode root) {
        sumSubtree(root);
        return totalTilt;
    }

    private int sumSubtree(TreeNode node) {
        if (node == null) return 0;
        int leftSum = sumSubtree(node.left);
        int rightSum = sumSubtree(node.right);
        totalTilt += Math.abs(leftSum - rightSum);
        return leftSum + rightSum + node.val;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

3. Find All Nodes Distance K from Target

Problem: Given a target node, find all nodes that are at distance K from the target node in a binary tree.
Example:

```makefile
Input:
       3
      / \
     5   1
    / \ / \
   6  2 0  8
     / \
    7   4

Target = 5, K = 2

Output: [7, 4, 1]
```

Java Code:

```java
public class NodesAtDistanceK {
    private Map<TreeNode, TreeNode> parentMap = new HashMap<>();

    public List<Integer> distanceK(TreeNode root, TreeNode target, int K) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;

        buildParentMap(root, null);
        Queue<TreeNode> queue = new LinkedList<>();
        Set<TreeNode> visited = new HashSet<>();
        queue.offer(target);
        visited.add(target);
        int distance = 0;

        while (!queue.isEmpty()) {
            if (distance == K) {
                for (TreeNode node : queue) {
                    result.add(node.val);
                }
                return result;
            }
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                if (node.left != null && visited.add(node.left)) queue.offer(node.left);
                if (node.right != null && visited.add(node.right))
queue.offer(node.right);
                TreeNode parent = parentMap.get(node);
                if (parent != null && visited.add(parent)) queue.offer(parent);
            }
            distance++;
        }
        return result;
    }

    private void buildParentMap(TreeNode node, TreeNode parent) {
        if (node == null) return;
        parentMap.put(node, parent);
        buildParentMap(node.left, node);
        buildParentMap(node.right, node);
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

4. Construct Binary Tree from Inorder and Postorder Traversal

Problem: Construct a binary tree given its inorder and postorder traversal arrays.
Example:

```makefile
Input:
Inorder = [9, 3, 15, 20, 7]
Postorder = [9, 15, 7, 20, 3]

Output:
       3
      / \
     9   20
```

```
        / \
       15  7
```

Java Code:

```java
public class ConstructTreeFromInPost {
    private int postIndex;

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        postIndex = postorder.length - 1;
        Map<Integer, Integer> inIndexMap = new HashMap<>();
        for (int i = 0; i < inorder.length; i++) {
            inIndexMap.put(inorder[i], i);
        }
        return buildTree(postorder, 0, inorder.length - 1, inIndexMap);
    }

    private TreeNode buildTree(int[] postorder, int inStart, int inEnd, Map<Integer,
Integer> inIndexMap) {
        if (inStart > inEnd) return null;
        TreeNode root = new TreeNode(postorder[postIndex--]);
        int inIndex = inIndexMap.get(root.val);
        root.right = buildTree(postorder, inIndex + 1, inEnd, inIndexMap);
        root.left = buildTree(postorder, inStart, inIndex - 1, inIndexMap);
        return root;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

5. Check if Binary Tree is a Subtree of Another Binary Tree

Problem: Determine if one binary tree is a subtree of another binary tree.
Example:

```yaml
Input:
Tree 1:
       3
      / \
     4   5
    / \
   1   2

Tree 2:
     4
    / \
   1   2

Output: true
```

Java Code:

```java
public class SubtreeOfAnotherTree {
    public boolean isSubtree(TreeNode s, TreeNode t) {
        if (s == null) return false;
        if (isSameTree(s, t)) return true;
        return isSubtree(s.left, t) || isSubtree(s.right, t);
    }

    private boolean isSameTree(TreeNode s, TreeNode t) {
        if (s == null && t == null) return true;
        if (s == null || t == null) return false;
        if (s.val != t.val) return false;
        return isSameTree(s.left, t.left) && isSameTree(s.right, t.right);
    }
}
```

Time Complexity: O(n * m)
Space Complexity: O(n)

---

6. Binary Tree Paths

Problem: Find all root-to-leaf paths in a binary tree.
Example:

```vbnet
Input:
       1
      / \
     2   3
      \
       5

Output: ["1->2->5", "1->3"]
```

Java Code:

```java
public class BinaryTreePaths {
    public List<String> binaryTreePaths(TreeNode root) {
        List<String> paths = new ArrayList<>();
        if (root != null) searchPaths(root, "", paths);
        return paths;
    }

    private void searchPaths(TreeNode root, String path, List<String> paths) {
        if (root.left == null && root.right == null) paths.add(path + root.val);
        if (root.left != null) searchPaths(root.left, path + root.val + "->", paths);
        if (root.right != null) searchPaths(root.right, path + root.val + "->", paths);
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

7. Mirror of Binary Tree

Problem: Find the mirror image of a binary tree (i.e., flip the tree horizontally).
Example:

```makefile
Input:
       4
      / \
     2   7
    / \ / \
   1  3 6  9

Output:
       4
      / \
     7   2
    / \ / \
   9  6 3  1
```

Java Code:

```java
public class MirrorBinaryTree {
    public TreeNode mirrorTree(TreeNode root) {
        if (root == null) return null;
        TreeNode left = mirrorTree(root.left);
        TreeNode right = mirrorTree(root.right);
        root.left = right;
        root.right = left;
        return root;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

8. Max Path Sum in Binary Tree

Problem: Find the maximum path sum (the sum of the maximum sum path between any two nodes) in a binary tree.
Example:

```makefile
Input:
       1
      / \
     2   3

Output: 6 (2 -> 1 -> 3)
```

Java Code:

java

```java
public class MaxPathSum {
    private int maxSum = Integer.MIN_VALUE;

    public int maxPathSum(TreeNode root) {
        maxGain(root);
        return maxSum;
    }

    private int maxGain(TreeNode node) {
        if (node == null) return 0;
        int leftGain = Math.max(maxGain(node.left), 0);
        int rightGain = Math.max(maxGain(node.right), 0);
        int priceNewPath = node.val + leftGain + rightGain;
        maxSum = Math.max(maxSum, priceNewPath);
        return node.val + Math.max(leftGain, rightGain);
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

---

9. Find the Closest Binary Search Tree Value

Problem: Given a BST and a target value, find the value in the tree that is closest to the target.
Example:

makefile

```makefile
Input:
       4
      / \
     2   5
    / \
   1   3

Target: 3.714286

Output: 4
```

Java Code:

java

```java
public class ClosestBSTValue {
    public int closestValue(TreeNode root, double target) {
        int closest = root.val;
        while (root != null) {
            if (Math.abs(target - root.val) < Math.abs(target - closest)) {
                closest = root.val;
            }
            root = target < root.val ? root.left : root.right;
        }
        return closest;
    }
}
```

Time Complexity: O(h) (where h is the height of the tree)
Space Complexity: O(1)

---

10. Flatten Binary Tree to Linked List

Problem: Flatten a binary tree to a linked list in-place using the right child pointer as the next pointer.
Example:

```rust
Input:
       1
      / \
     2   5
    / \   \
   3   4   6

Output:
1 -> 2 -> 3 -> 4 -> 5 -> 6
```

Java Code:

```java
public class FlattenBinaryTree {
    public void flatten(TreeNode root) {
        TreeNode cur = root;
        while (cur != null) {
            if (cur.left != null) {
                TreeNode pre = cur.left;
                while (pre.right != null) {
                    pre = pre.right;
                }
                pre.right = cur.right;
                cur.right = cur.left;
                cur.left = null;
            }
            cur = cur.right;
        }
    }
}
```

Time Complexity: O(n)
Space Complexity: O(1)

These solutions cover all the remaining problems with their respective examples and optimized Java code.

ChatGPT can make mistakes. Check important info.