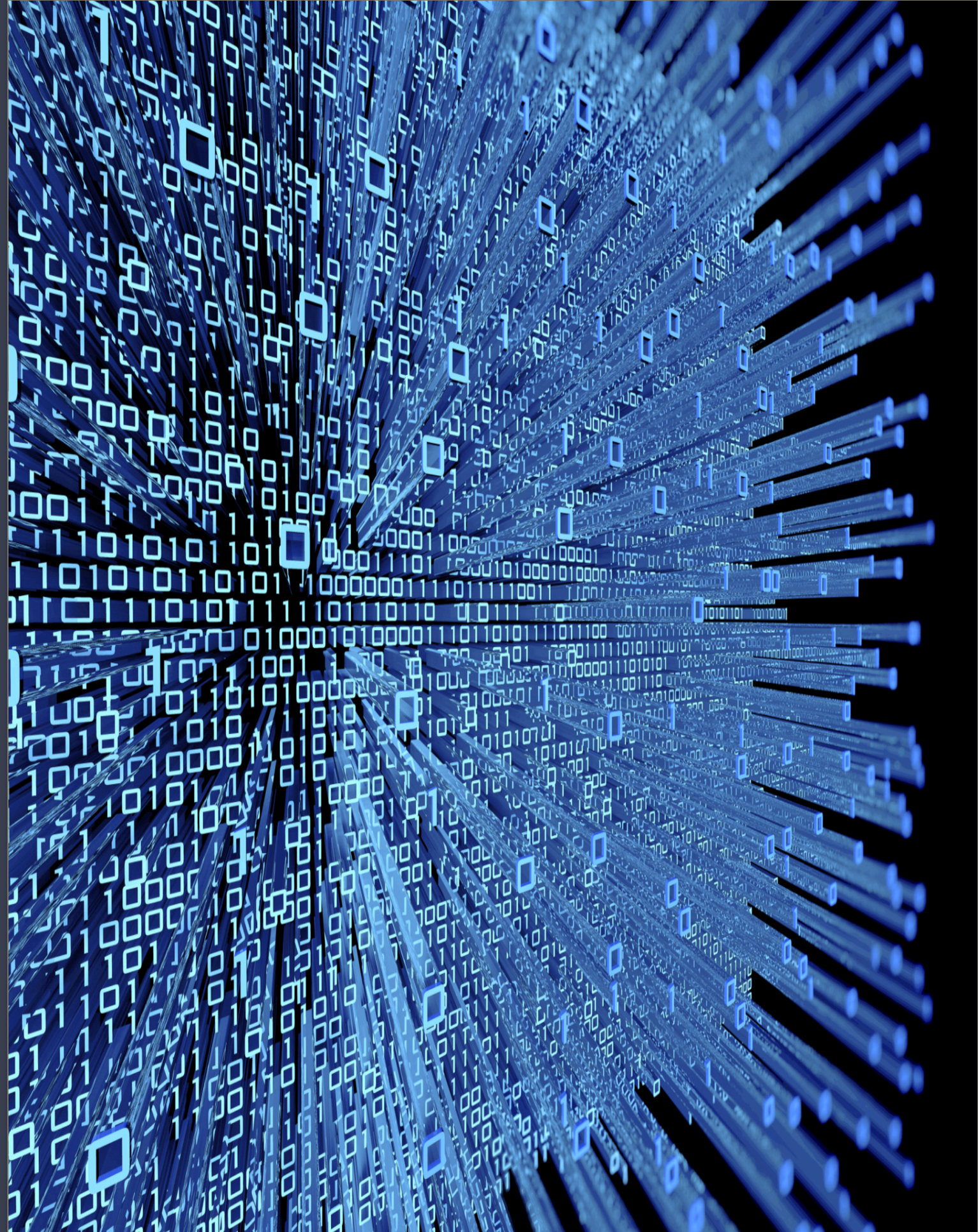
Sunil Yadav

TOP INTERVIEW QUESTIONS

EASY COLLECTIONS

TREE



INTRODUCTION

Tree is slightly more complex than linked list, because the latter is a linear data structure while the former is not. Tree problems can be solved either **breadth-first** or **depth-first**. We have one problem here which is great for practicing breadth-first traversal.

We recommend: Maximum Depth of Binary Tree, Validate Binary Search Tree, Binary Tree Level Order Traversal and Convert Sorted Array to Binary Search Tree.

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 root node down to the farthest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its depth = 3.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int maxDepth(TreeNode root) {
    if(root==null){
        return 0;
    }
    return 1 +
Math.max(maxDepth(root.left),maxDepth(root.right));
    }

}
```

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return isBST(root, null, null);
    }

    private boolean isBST(TreeNode root,Integer MIN, Integer MAX){
        if(root == null){
            return true;
        }
        if(MIN !=null && root.val <= MIN)
            return false;
        if(MAX !=null && root.val >= MAX)
            return false;
        if(!isBST(root.left, MIN, root.val))
            return false;
        if(!isBST(root.right, root.val,MAX))
            return false;
        return true;
        }
}
```

Validate Binary Search Tree

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

```
    2
   / \
  1   3
```

Input: [2,1,3]
Output: true

Example 2:

```
    5
   / \
  1   4
     / \
    3   6
```

Input: [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's value is 4.

Symmetric Tree

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree `[1,2,2,3,4,4,3]` is symmetric:

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

But the following `[1,2,2,null,3,null,3]` is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

Follow up: Solve it both recursively and iteratively.

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if(root==null){
            return true;
        }
        if(root.left==null && root.right==null){
            return true;
        }
        if((root.left!=null && root.right==null) ||(root.left==null && root.right!
=null)){
            return false;
        }

        if((root.left.val != root.right.val)) {
            return false;
        }
       return is(root.left,root.right);
    }
    private boolean is(TreeNode l,TreeNode r){
        if(l==null && r==null){
            return true;
        }
        if((l == null && r!=null)||(l != null && r==null)){
            return false;
        }
        if(l.val!=r.val){
            return false;
        }
        return is(l.left,r.right) && is(l.right,r.left);
    }
}
```

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        if(root==null){
            return new ArrayList<>();
        }
        List<List<Integer>> res = new ArrayList<>();
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        while(true){
            int n = queue.size();
            if(n==0){
                break;
            }
            List<Integer> nP = new ArrayList<>();
            while(n > 0){
                TreeNode xyz = queue.poll();
                if(xyz!=null){
                    nP.add(xyz.val);
                    if(xyz.left!=null){
                        queue.add(xyz.left);
                    }
                    if(xyz.right!=null){
                        queue.add(xyz.right);
                    }
                }
                n--;
            }
            res.add(nP);
        }
        return res;
    }


}
```

Binary Tree Level Order Traversal

Given a binary tree, return the *level order* traversal of its nodes' values. (ie, from left to right, level by level).

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]
]
```

Convert Sorted Array to Binary Search Tree

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

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.

Example:

```
Given the sorted array: [-10,-3,0,5,9],
```

```
One possible answer is: [0,-3,9,-10,null,5], which represents the following height balanced BST:
```

```
      0
     / \
   -3   9
   /   /
 -10   5
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        return buildBST(nums,0,nums.length-1);
    }
    private TreeNode buildBST(int[] nums,int startIndex, int endIndex){
        if(startIndex > endIndex){
            return null;
        }
        int midIndex = (endIndex+startIndex)/2;
        TreeNode root = new TreeNode(nums[midIndex]);
        root.left = buildBST(nums,startIndex,midIndex-1);
        root.right = buildBST(nums,midIndex+1,endIndex);
        return root;
    }
}
```