Binary Tree Diameter DSA

```
Give DSA notes along with this code: class Solution {
    private int maxPath = 0;

    public int diameterOfBinaryTree(TreeNode root) {
        calculateHeight(root);
        return maxPath;
    }

    private int calculateHeight(TreeNode node) {
        if (node == null) return 0;

        // Recursively calculate heights of left and right
subtrees
        int leftHeight = calculateHeight(node.left);
        int rightHeight = calculateHeight(node.right);

        // Update the maximum diameter
        maxPath = Math.max(maxPath, leftHeight +
rightHeight);

        // Return the height of the current subtree
        return 1 + Math.max(leftHeight, rightHeight);
    }
}
```

The code provided calculates the **diameter of a binary tree**. The diameter is the longest path between any two nodes in a tree, which may or may not pass through the root.

DSA Notes: Binary Tree Diameter

Problem Overview:

- **Diameter of Binary Tree** is defined as the number of nodes on the longest path between any two leaves in the tree.
- The diameter may pass through the root or may not.

Solution Approach:

1. **Height of a Binary Tree:**

- The height of a node in a binary tree is defined as the number of edges on the longest path from that node to a leaf node.
- To compute the diameter, we need to calculate the height of the left and right subtrees for each node and check if the sum of those heights (which would give the number of edges between the leftmost and rightmost nodes through the current node) is greater than the previously recorded diameter.

2. **Recursive Approach:**
   - We use recursion to traverse through the tree. For each node, we calculate the height of its left and right subtrees.
   - The height of the node will be `1 + max(leftHeight, rightHeight)`, and the diameter of the tree can be updated as `max(maxDiameter, leftHeight + rightHeight)`.
3. **Base Case:**
   - If the node is `null`, the height is `0`, as there are no nodes in that subtree.
4. **Key Variables:**
   - **maxPath**: This variable stores the maximum diameter found during the traversal.

Code Walkthrough:

```java
class Solution {
    private int maxPath = 0;  // This stores the maximum diameter encountered

    // Main function to compute the diameter
    public int diameterOfBinaryTree(TreeNode root) {
        calculateHeight(root);  // Start recursive computation
        return maxPath;         // Return the maximum diameter found
    }

    // Helper function to compute height and update the diameter
    private int calculateHeight(TreeNode node) {
        if (node == null) return 0;  // Base case: height of a null node is 0

        // Recursively calculate the height of the left and right subtrees
        int leftHeight = calculateHeight(node.left);
        int rightHeight = calculateHeight(node.right);
        // Update the maximum diameter found so far
        maxPath = Math.max(maxPath, leftHeight + rightHeight);

        // Return the height of the current node's subtree
        return 1 + Math.max(leftHeight, rightHeight);
    }
}
```

Explanation:

1. `diameterOfBinaryTree(TreeNode root)`:
    - This is the main function that starts the recursive calculation. It calls `calculateHeight(root)` to begin traversing the tree.
2. `calculateHeight(TreeNode node)`:
    - This recursive function computes the height of the tree rooted at `node`.
    - For each node, the heights of the left and right subtrees are calculated, and the maximum diameter (`maxPath`) is updated if the sum of the left and right heights exceeds the current `maxPath`.
3. **Recursive Depth and Complexity**:
    - The time complexity of this approach is **O(n)**, where `n` is the number of nodes in the tree. This is because we visit each node once.
    - The space complexity is **O(h)**, where `h` is the height of the tree, due to the recursive call stack.

Example:

Consider the binary tree:

```markdown

        1
       / \
      2   3
     / \
    4   5
```

- The height of node 1 is `3` (through nodes 2, 4, and 5).

- The longest path is from node 4 to node 5, passing through node 2, so the diameter is `3` (edges between nodes 4 and 5).

Conclusion:

This approach effectively computes the diameter by leveraging a post-order traversal (left-right-root) to calculate the height of each subtree and update the maximum diameter.