# Traverse vs Divide & Conquer

> - They are both Recursion Algorithm
> - Result in `parameter` vs Result in `return value`
> - Top down vs Bottom up

---

## 求二叉树的深度

```
/*
    分别用traversal和分治法求二叉树的深度
*/
//traversal
class Solution {
    int max;
    public int maxDepth(TreeNode root) {
        max = 0;
        traversal(root, 0);
        return max;
    }

    public void traversal(TreeNode node, int curr){
        if(node == null) return;
        max = Math.max(max, curr+1);
        traversal(node.left, curr+1);
        traversal(node.right, curr+1);
    }
}
//divide & conqure
class Solution {
    public int maxDepth(TreeNode root) {
        if(root == null) return 0;
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        return Math.max(left, right) + 1;
    }
}
```

## 求解二叉树的所有路径

## 257. Binary Tree Paths

Easy 1407 90 Add to List Share

Given a binary tree, return all root-to-leaf paths.

**Note:** A leaf is a node with no children.

**Example:**

**Input:**

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

**Explanation:** All root-to-leaf paths are: 1->2->5, 1->3

```java
class Solution {
    public List<String> binaryTreePaths(TreeNode root) {
        List<String> res = new ArrayList<>();
        if(root == null) return res;
        List<String> left = binaryTreePaths(root.left);
        List<String> right = binaryTreePaths(root.right);
        if(left.size() == 0 && right.size() == 0){
            res.add(root.val+"");
        }else {
            for(String s : left){
                res.add(root.val+"->" + s);
            }

            for(String s : right){
                res.add(root.val+"->" + s);
            }
        }
        return res;
    }
}
```