```swift
import Foundation
/// **剑指 Offer 37. 序列化二叉树**
/// 请实现两个函数，分别用来序列化和反序列化二叉树。
/// 你需要设计一个算法来实现二叉树的序列化与反序列化。这
/// 里不限定你的序列 / 反序列化算法执行逻辑，你只需要保证一个二叉树可以被序列化
/// 为一个字符串并且将这个字符串反序列化为原始的树结构。
/// 提示：输入输出格式与 LeetCode 目前使用的方式一致，详情请参阅 LeetCode
/// 序列化二叉树的格式。你并非必须采取这种方式，你也可以采用其他的方法解决这个问题。
///
/// 输入：root = [1, 2, 3, null, null, 4, 5]
/// 输出：[1, 2, 3, null, null, 4, 5]
class Solution {
    // 序列化二叉树为字符串
    func serialize(_ root: TreeNode?) -> String {
        // 如果根节点为空，则直接返回空列表的字符串表示
        guard let root = root else { return "[]" }
        // 使用队列辅助进行层序遍历
        var queue = [TreeNode?]()
        queue.append(root)
        var res = [String]()
        while !queue.isEmpty {
            if let node = queue.removeFirst() {
                // 将节点值添加到结果数组中
                res.append(String(node.val))
                // 将左右子节点加入队列中，继续遍历
                queue.append(node.left)
                queue.append(node.right)
            } else {
                // 如果节点为空，添加 "null" 到结果数组中
                res.append("null")
            }
        }
        // 返回序列化后的二叉树字符串
        return "[" + res.joined(separator: ",") + "]"
    }

    // 反序列化字符串为二叉树
    func deserialize(_ data: String) -> TreeNode? {
        // 将字符串转换为数组，并去掉开头和结尾的方括号
        let vals = data[data.index(after:
         data.startIndex)..<data.index(before: data.endIndex)]
            .split(separator: ",")
            .map { String($0) }
        // 如果数组为空，则返回空节点
        guard !vals.isEmpty else { return nil }
        // 使用索引 i 记录当前处理到的节点值在 vals 数组中的位置
        var i = 1
        // 创建根节点，并加入队列
```

```
        let root = TreeNode(Int(vals[0])!)
        var queue = [TreeNode]()
        queue.append(root)
        while !queue.isEmpty {
            if vals[i] != "null" {
                // 如果左子节点值不为空，创建左子节点并加入队列
                let left = TreeNode(Int(vals[i])!)
                queue[0].left = left
                queue.append(left)
            }
            i += 1
            if vals[i] != "null" {
                // 如果右子节点值不为空，创建右子节点并加入队列
                let right = TreeNode(Int(vals[i])!)
                queue[0].right = right
                queue.append(right)
            }
            i += 1
            // 移除当前节点，继续处理下一个节点
            queue.removeFirst()
        }
        // 返回反序列化后的二叉树根节点
        return root
    }
}

public class TreeNode {
    public var val: Int
    public var left: TreeNode?
    public var right: TreeNode?
    public init() { self.val = 0; self.left = nil; self.right = nil; }
    public init(_ val: Int) { self.val = val; self.left = nil; self.right =
     nil; }
    public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
        self.val = val
        self.left = left
        self.right = right
    }
}
```