```swift
import Foundation
// 剑指 Offer 26. 树的子结构(理解并背诵)
// 输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)
//
// B是A的子结构， 即 A中有出现和B相同的结构和节点值。
//
// 例如:
// 给定的树 A:
//      3
//     / \
//    4   5
//   / \
// 1   2
// 给定的树 B:
//    4
//   /
// 1
// 返回 true，因为 B 与 A 的一个子树拥有相同的结构和节点值。
class Solution {
    func isSubStructure(_ A: TreeNode?, _ B: TreeNode?) -> Bool {
        if (B == nil) || (A == nil) { return false }
        /// B为A的子结构有3种情况,满足任意一种即可:
        ///  1.B的子结构起点为A的根节点,此时结果为recur(A,B)
        ///
             2.B的子结构起点隐藏在A的左子树中,而不是直接为A的根节点,此时结果为isSubStructure(A
             .left, B)
        ///  3.B的子结构起点隐藏在A的右子树中,此时结果为isSubStructure(A.right, B)
        return recur(A, B) || isSubStructure(A?.left, B) ||
         isSubStructure(A?.right, B)
    }

    /// 判断B是否为A的子结构,其中B子结构的起点为A的根节点
    func recur(_ A: TreeNode?, _ B: TreeNode?) -> Bool {
        /// 若B走完了,说明查找完毕,B为A的子结构
        if B == nil { return true }
        /// 若B不为空并且A为空或者A与B的值不相等,直接可以判断B不是A的子结构
        if A == nil || (A?.val != B?.val) {
            return false
        }
        /// 当A与B当前节点值相等,若要判断B为A的子结构
        /// 还需要判断B的左子树是否为A左子树的子结构 && B的右子树是否为A右子树的子结构
        /// 若两者都满足就说明B是A的子结构,并且该子结构以A根节点为起点
        return recur(A?.left, B?.left) && recur(A?.right, B?.right)
    }
}
```