```
                    e1P.right = e2;
                    e2P.left = e1;
                } else { // 情况 14
                    e1.left = e2L;
                    e1.right = e2R;
                    e2.left = e1L;
                    e2.right = e1R;
                    e1P.right = e2;
                    e2P.right = e1;
                }
            }
        }
    }
    return head;
}
```

判断 t1 树是否包含 t2 树全部的拓扑结构

【题目】

给定彼此独立的两棵树头节点分别为 t1 和 t2，判断 t1 树是否包含 t2 树全部的拓扑结构。例如，图 3-34 所示的 t1 树和图 3-35 所示的 t2 树。

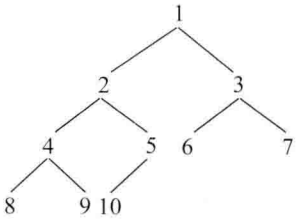

图 3-34

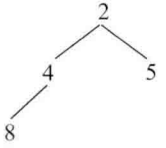

图 3-35

t1 树包含 t2 树全部的拓扑结构，所以返回 true。

【难度】

士 ★☆☆☆

【解答】

如果 t1 中某棵子树头节点的值与 t2 头节点的值一样，则从这两个头节点开始匹配，匹配的每一步都让 t1 上的节点跟着 t2 的先序遍历移动，每移动一步，都检查 t1 的当前节点

是否与 t2 当前节点的值一样。比如，题目中的例子，t1 中的节点 2 与 t2 中的节点 2 匹配，然后 t1 跟着 t2 向左，发现 t1 中的节点 4 与 t2 中的节点 4 匹配，t1 跟着 t2 继续向左，发现 t1 中的节点 8 与 t2 中的节点 8 匹配，此时 t2 回到 t2 中的节点 2，t1 也回到 t1 中的节点 2，然后 t1 跟着 t2 向右，发现 t1 中的节点 5 与 t2 中的节点 5 匹配。t2 匹配完毕，结果返回 true。如果匹配的过程中发现有不匹配的情况，直接返回 false，说明 t1 的当前子树从头节点开始，无法与 t2 匹配，那么再去寻找 t1 的下一棵子树。t1 的每棵子树上都有可能匹配出 t2，所以都要检查一遍。

所以，如果 t1 的节点数为 N，t2 的节点数为 M，该方法的时间复杂度为 $O(N \times M)$。

具体过程请参看如下代码中的 contains 方法，

```
public class Node {
        public int value;
        public Node left;
        public Node right;

        public Node(int data) {
                this.value = data;
        }
}

public boolean contains(Node t1, Node t2) {
        return check(t1, t2) || contains(t1.left, t2) || contains(t1.right, t2);
}

public boolean check(Node h, Node t2) {
        if (t2 == null) {
                return true;
        }
        if (h == null || h.value != t2.value) {
                return false;
        }
        return check(h.left, t2.left) && check(h.right, t2.right);
}
```

遍历每一个节点尝试找出相同的头节点来检查

检查结构是否一致

判断 t1 树中是否有与 t2 树拓扑结构完全相同的子树

【题目】

给定彼此独立的两棵树头节点分别为 t1 和 t2，判断 t1 中是否有与 t2 树拓扑结构完全相同的子树。

例如，图 3-36 所示的 t1 树和图 3-37 所示 t2 树。