# 题目

检查子树。你有两棵非常大的二叉树：T1，有几万个节点；T2，有几万个节点。设计一个算法，判断 T2 是否为 T1 的子树。

如果 T1 有这么一个节点 n，其子树与 T2 一模一样，则 T2 为 T1 的子树，也就是说，从节点 n 处把树砍断，得到的树与 T2 完全相同。

注意：此题相对书上原题略有改动。

示例1:

输入：t1 = [1, 2, 3], t2 = [2]

输出：true

示例2:

输入：t1 = [1, null, 2, 4], t2 = [3, 2]

输出：false

提示：

树的节点数目范围为[0, 20000]。

# 分析

## 方法一：递归法（推荐）

**思路：**

使用递归的方法判断。

**代码：**

class Solution {

public:

    bool isSubtree(TreeNode\* root, TreeNode\* subRoot) {

        if(!root)   return false;

        return isSubtree(root->left,subRoot) || isSame(root,subRoot) || isSubtree(root->right,subRoot);

    }

    bool isSame(TreeNode \*p,TreeNode \*q)

    {

        if(nullptr==p && nullptr==q)    return true;

        if(nullptr==p || nullptr==q)    return false;

       return p->val==q->val && isSame(p->left,q->left) && isSame(p->right,q->right);

    }

};

或：

bool isSameTree(TreeNode\* s, TreeNode\* t) {

if (!s && !t) return true;

if (!s || !t) return false;

if (s->val != t->val) return false;

return isSameTree(s->left, t->left) && isSameTree(s->right, t->right);

}

分析：

时间复杂度：O(m \* n)

空间复杂度：O(h1 + h2)（递归栈开销）

## 方法二：字符串匹配+序列化（推荐）

思路：

我们将两棵树（主树 root 和子树 subRoot）序列化为字符串，然后通过哈希（如 std::unordered\_set）或字符串匹配来判断是否存在完全相同的子树。

核心步骤

1、将整棵树 root 序列化为一棵“唯一的字符串”形式（带有空节点标记，防止结构歧义）；

2、将 subRoot 也序列化为同样格式的字符串；

3、使用字符串匹配或哈希判断 subRoot 的序列是否在 root 中出现过。

代码：

class Solution {

public:

// 后序遍历序列化（包含 null 标记）

string serialize(TreeNode\* node) {

if (!node) return "#";

return "(" + serialize(node->left) + ")" + to\_string(node->val) + "(" + serialize(node->right) + ")";

}

bool checkSubTree(TreeNode\* t1, TreeNode\* t2) {

string sRoot = serialize(t1);

string sSub = serialize(t2);

return sRoot.find(sSub) != string::npos;

}

};

**为什么要加括号/#？**

不加会出现结构歧义，如 12 和 1-2；

加括号或特殊符号可保证树结构唯一性。

**序列化函数常见错误：**

string serialize(TreeNode\* node) {

if (!node) return "#"; // 空节点用 # 占位

return to\_string(node->val) + "," + serialize(node->left) + "," + serialize(node->right);

}

root = [12] 序列化后是 "12,#,#"

subRoot = [2] 序列化后是 "2,#,#"

因为字符串中包含 "2"，所以用 find 会误判为 true。

你的序列化已经加了逗号分隔符，改成 "," + to\_string(node->val) + "," 等形式，避免子串误判。

修改序列化函数，确保：

每个节点值左右都有逗号分隔

空节点也用 # 标记并且前后都有逗号

**正确的函数：**

string serialize(TreeNode\* node) {

if (!node) return ",#";

return "," + to\_string(node->val) + serialize(node->left) + serialize(node->right);

}

root = [12] 序列化后是 ",12,#,#"

subRoot = [2] 序列化后是 ",2,#,#"

**分析：**

时间复杂度：O(m + n)，m 为 root 节点数，n 为 subRoot 节点数（遍历 + 字符串匹配）

空间复杂度：O(m + n)，序列化后的字符串长度