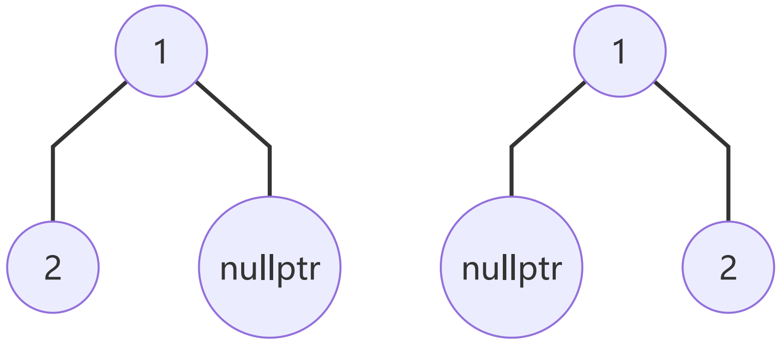
1

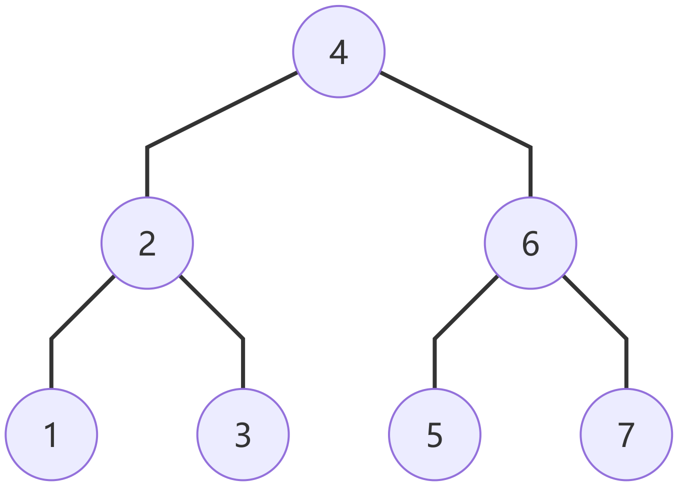
不能，反例为：



这两棵二叉树的前序遍历、后序遍历序列都是相同的.

2

考虑这样一颗二叉树：



取路径 $S_2 = \{4, 6, 7\}$，有$5 \in S_1, 4 \in S_2$，此为表达式 1 的反例.

取路径 $S_2 = 4, 2, 1$，有 $3 \in S_3, 4 \in S_2$，此为表达式 2 的反例.

表达式 3 总是正确的，因为 a, c 的公共祖先 p 必然在路径上，且 a 在 p 的左子树中，c 在 p 的右子树中，故必有 $a < c$.

3

定义左前序遍历为按“左子树—根—右子树”的顺序遍历二叉树，右前序遍历为按“右子树—根—左子树”的顺序遍历二叉树，类似可定义左中序遍历、右中序遍历.

显然，一颗对称二叉树的左前序遍历序列等于其右前序遍历序列，其左中序遍历序列也等于其右中序遍历序列. 另一方面，如果一颗二叉树的左前序遍历序列等于右前序遍历序列，左中序遍历序列也等于右中序遍历序列，那它一定是对称的（题目未严格定义对称，我也无法严格证明）.

因此，只需作四次遍历并比较所得序列即可，其算法实现如下：

```
bool isSymmetric(Node<T>* root){
    return preorder(root) && inorder(root);
}

void preorder(Node<T>* root){
    std::stack<Node<T>*> stkl, stkr;
    stkl.push(nullptr); stkr.push(nullptr);
    Node<T>* l = root;
    Node<T>* r = root;
    while (l && r){
        if (l->data != r->data)
            return false;

        if (l->right)
            stkl.push(l->right);
        if (l->left)
            l = l->left;
        else{
            l = stkl.top();
            stkl.pop();
        }

        if (r->left)
            stkr.push(r->left);
        if (r->right)
            r = r->right;
        else{
            r = stkr.top();
            stkr.pop();
        }
    }
    return true;
}

void inorder(Node<T>* root){
    std::stack<Node<T>*> stkl, stkr;
    Node<T>* l = root;
    Node<T>* r = root;
    while ((l && r) || !stk.empty()){
        while (l){
            stkl.push(l);
            l = l->left;
        }
        while (r){
            stkr.push(r);
            r = r->right;
        }

        l = stkl.top(); stkl.pop();
        r = stkr.top(); stkr.pop();

        if (l->data != r->data)
            return false;
```

```
        l = l->right;
        r = r->left;
    }
    return true;
}
```

这个算法的时空复杂度都是 $O(N)$，其中 N 为总结点数.

4

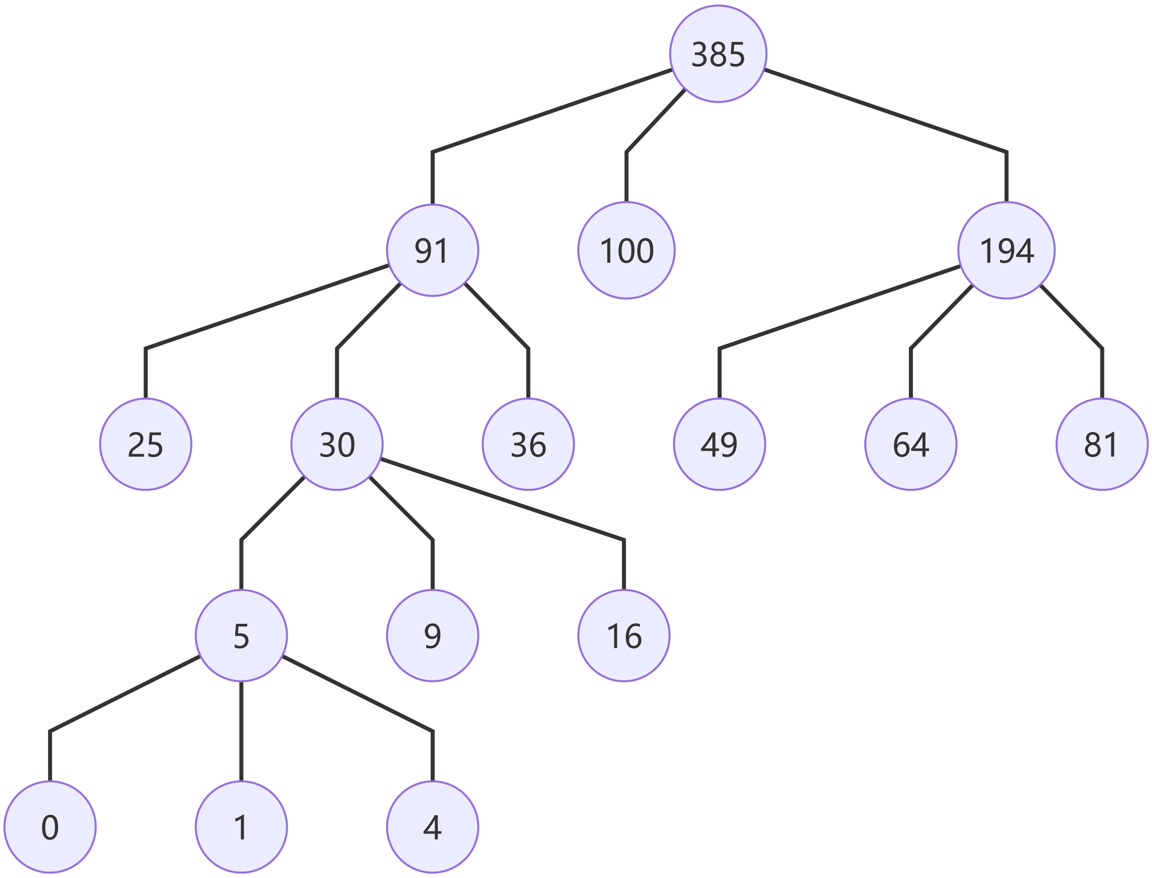
按堆的定义进行检查即可：

```
bool isHeap(int* arr, int m){
    for (int i = m-1; i; --i)
        if (arr[(i-1)/2] < arr[i])
            return false;
    return true;
}
```

这个算法的时间是 $O(m)$ 的，空间是 $O(1)$ 的.

5

设 $t=(n-1)\%(k-1), r=(t==0)?0:(k-1-t)$，补充 r 个权值为 0 的结点，并像构造 Huffman 树一样不断合并 k 个结点再加入权值等于其权值和的新结点，直到只剩一个结点即可.



6

先将 p, q 中较深的结点回溯到较浅结点所在的层数，再共同回溯两者，直到回到同一个结点，这个结点即为最近公共祖先. 这个算法的时间复杂度是 $O(H)$，空间复杂度是 $O(1)$，其中 H 为二叉树的高度.

```
int calcDepth(Node<T>* root){
    int res = -1;
    for (; root; root = root->parent, ++res);
    return res;
}

Node<T>* LCA(Node<T>* rt, Node<T>* p, Node<T>* q){
    int dp = calcDepth(p), dq = calcDepth(q);
    int dif = dp-dq;
    if (dif > 0){
        while (dif--) p = p.parent;
    }
    else{
        dif = -dif;
        while (dif--) q = q.parent;
    }
    for (; p != q; p = p.parent, q = q.parent);
    return p;
}
```