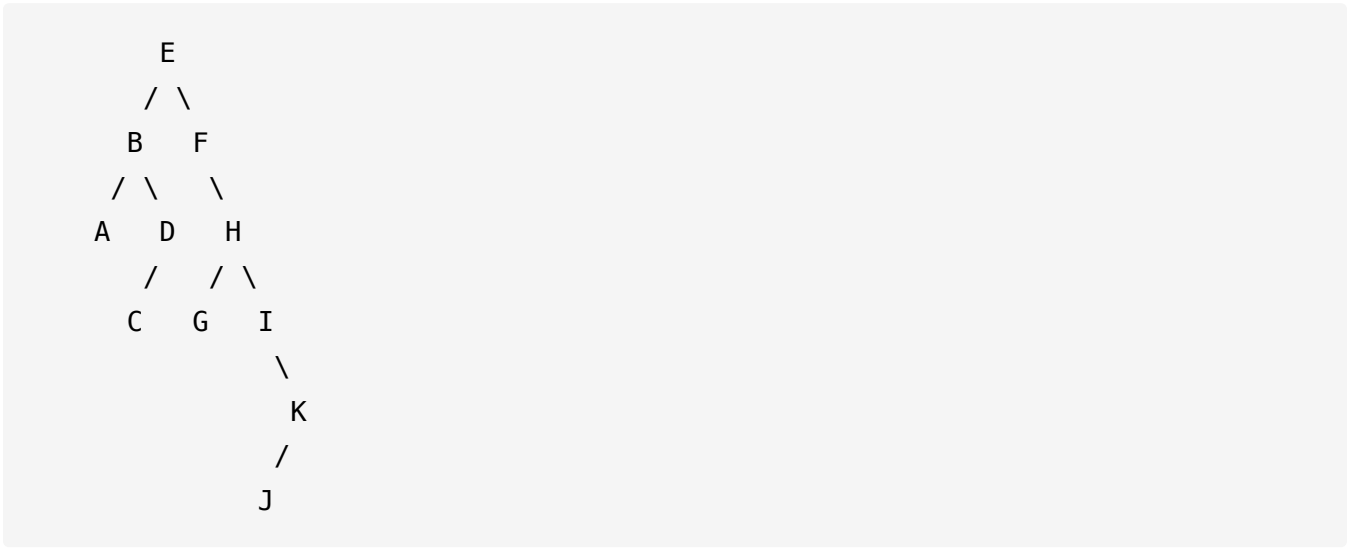
Homework4

23336003 陈政宇

1.

先序遍历：EBADCFHGIKJ

中序遍历：ABCDEFGHIJK



2.

分析和计算过程

1. **满二叉树的性质**：
    - 满二叉树是一种特殊的二叉树，其中每个节点要么是叶子节点，要么有两个子节点。
    - 满二叉树的节点总数 n 与其高度 h 之间的关系为：

$$n = 2^{h+1} - 1$$

    - 满二叉树的叶子节点数 L 与其高度 h 之间的关系为：

$$L = 2^h$$

2. **节点个数为 20~40 之间的素数**：

- 在 20 到 40 之间的素数有：23、29、31、37

3. **计算每个素数对应的叶子节点数**：
   - 对于每个素数 n，我们需要找到满足 $n = 2^{h+1} - 1$ 的高度 h，然后计算叶子节点数 $L = 2^h$。

计算过程

1. **节点数 $n = 23$**：
   - $2^{h+1} - 1 = 23$
   - $2^{h+1} = 24$
   - $h + 1 = \log_2 24 \approx 4.58$（不是整数，不能构成满二叉树）
2. **节点数 $n = 29$**：
   - $2^{h+1} - 1 = 29$
   - $2^{h+1} = 30$
   - $h + 1 = \log_2 30 \approx 4.91$（不是整数，不能构成满二叉树）
3. **节点数 $n = 31$**：
   - $2^{h+1} - 1 = 31$
   - $2^{h+1} = 32$
   - $h + 1 = \log_2 32 = 5$
   - $h = 4$
   - 叶子节点数 $L = 2^h = 2^4 = 16$
4. **节点数 $n = 37$**：
   - $2^{h+1} - 1 = 37$
   - $2^{h+1} = 38$
   - $h + 1 = \log_2 38 \approx 5.25$（不是整数，不能构成满二叉树）

结论

在 20 到 40 之间的素数中，只有 31 可以构成满二叉树。对于节点数为 31 的满二叉树，其叶子节点数为 16。

3.

```
void printAncestors(TreeNode* root, int x) {
    if (!root) return;

    std::stack<TreeNode*> stack;
    TreeNode* lastVisited = nullptr;
    TreeNode* current = root;

    while (!stack.empty() || current) {
        if (current) {
            stack.push(current);
            current = current->left;
        } else {
            TreeNode* peekNode = stack.top();
            if (peekNode->right && lastVisited != peekNode->right) {
                current = peekNode->right;
            } else {
                if (peekNode->value == x) {
                    // Print all ancestors
                    std::stack<TreeNode*> tempStack = stack;
                    tempStack.pop(); // Remove the node itself
                    while (!tempStack.empty()) {
                        std::cout << tempStack.top()->value << " ";
                        tempStack.pop();
                    }
                    std::cout << std::endl;
                    return;
                }
                lastVisited = stack.top();
                stack.pop();
            }
        }
    }
}
```

4.

```
bool isCompleteBinaryTree(TreeNode* root) {
    if (!root) return true;

    std::queue<TreeNode*> q;
    q.push(root);
    bool mustBeLeaf = false;

    while (!q.empty()) {
        TreeNode* current = q.front();
        q.pop();

        if (current->left) {
            if (mustBeLeaf) return false;
            q.push(current->left);
        } else {
            mustBeLeaf = true;
        }

        if (current->right) {
            if (mustBeLeaf) return false;
            q.push(current->right);
        } else {
            mustBeLeaf = true;
        }
    }

    return true;
}
```