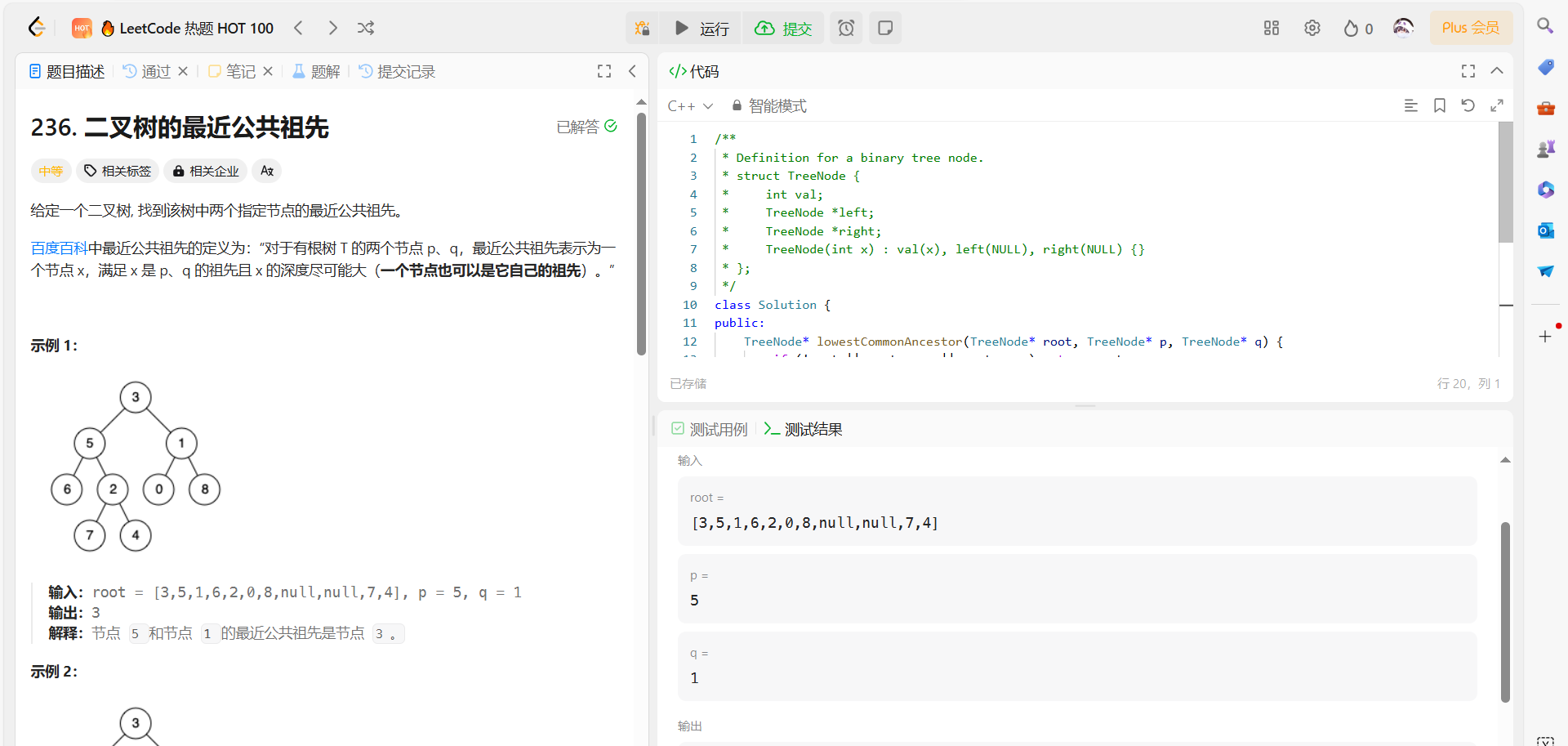
题目：



解题思路一：

递归法：

每次从当前节点开始，如果当前节点是空则返回空，如果是p或者q中的一个，返回p或者q;递归地在左子树和右子树中查找p和q,如果p和q分别在左右子树中，返回当前节点;否则返回非空子树的结果。

class Solution {

public:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

// 如果当前节点为空，返回空

if (!root) return nullptr;

// 如果当前节点是p或q中的一个，返回当前节点

if (root == p || root == q) return root;

// 递归在左子树中查找p和q

TreeNode\* left = lowestCommonAncestor(root->left, p, q);

// 递归在右子树中查找p和q

TreeNode\* right = lowestCommonAncestor(root->right, p, q);

// 如果p和q分别在左右子树中，返回当前节点

if (left && right) return root;

// 否则返回非空子树的结果

return left ? left : right;

}

};

解题思路二：

哈希表存储父亲节点：

使用DFS将从根节点开始的每个节点的父亲节点存入在哈希表中，如parent[p]中存储的是p的父亲节点，将它从它开始，依次存入它和它的父亲节点，为了避免值重复所以使用集合，从另一个节点q开始，如果它存在于p的祖先节点集合中，返回这个节点，如果不存在，q变成q的父亲节点，直到q为空跳出循环

class Solution {

// 哈希表存储每个节点的父节点

unordered\_map<TreeNode\*, TreeNode\*> parent;

// 使用深度优先搜索来填充父节点哈希表

void dfs(TreeNode\* root) {

if (root->left) {

parent[root->left] = root;

dfs(root->left);

}

if (root->right) {

parent[root->right] = root;

dfs(root->right);

}

}

public:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

// 根节点的父节点为nullptr

parent[root] = nullptr;

// 填充父节点哈希表

dfs(root);

// 集合存储p节点的所有祖先

unordered\_set<TreeNode\*> ancestors;

while (p) {

ancestors.insert(p);

p = parent[p];

}

// 从q节点开始，找到第一个在p的祖先集合中的节点

while (q) {

if (ancestors.count(q)) return q;

q = parent[q];

}

return nullptr;

}

};