

题目：

236. 二叉树的最近公共祖先

给定一个二叉树，找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个节点 p、q，最近公共祖先表示为一个节点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

示例 1:

输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1  
输出: 3  
解释: 节点 5 和节点 1 的最近公共祖先是节点 3。

```
1 /**  
2  * Definition for a binary tree node.  
3  * struct TreeNode {  
4  *     int val;  
5  *     TreeNode *left;  
6  *     TreeNode *right;  
7  *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}  
8  * };  
9  */  
10 class Solution {  
11 public:  
12     TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {  
13         // ...  
14     }  
15 };
```

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亲节点
    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) {
        // 根节点的父亲节点为 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.find(q) != ancestors.end())
                return q;
            q = parent[q];
        }
    }
};
```

```
        if (ancestors.count(q)) return q;
        q = parent[q];
    }

    return nullptr;
};
```