# 题目

给定一棵二叉树的根节点root以及两个整数p和q，返回该二叉树中值为p的结点与值为q的结点间的距离 。

两个结点间的 距离 就是从一个结点到另一个结点的路径上边的数目。

示例 1：



输入：root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 0

输出：3

解释：在 5 和 0 之间有 3 条边：5-3-1-0

示例 2：



输入：root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 7

输出：2

解释：在 5 和 7 之间有 2 条边：5-2-7

示例 3：



输入：root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 5

输出：0

解释：一个结点与它本身之间的距离为 0

提示：

树中结点个数的范围在 [1, 104].

0 <= Node.val <= 109

树中所有结点的值都是唯一的.

p 和q 是树中结点的值.

# 分析

算法1

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

typedef struct TreeNode TreeNode;

// function declaration

TreeNode\* lowestCommonAncestor(TreeNode\*, int, int);

void preOrder(TreeNode\*, int, int, int, int\*);

// 总的想法：把原问题转化成求LCA（Lowest Common Ancestor）问题

int findDistance(TreeNode\* root, int p, int q) {

TreeNode\* lca = lowestCommonAncestor(root, p, q);

int ans = 0;

preOrder(lca, 0, p, q, &ans);

return ans;

}

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

if (!root || root->val == p || root->val == q) return root;

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

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

return l && r ? root : l ? l : r;

}

void preOrder(TreeNode\* root, int d, int p, int q, int\* ans) {

if (!root) return;

if (root->val == p || root->val == q) \*ans += d;

preOrder(root->left, d + 1, p, q, ans);

preOrder(root->right, d + 1, p, q, ans);

}

算法2

// 图遍历

class Solution {

public:

int findDistance(TreeNode\* root, int p, int q) {

// step1: buid the undirected graph

buildGraph(root, nullptr);

// printGraph();

// 设 p为 start, q == goal 当然反过来也不是不可以....

// bfs, dfs search 都可以 看个人喜好，我比较偏爱dfs

// 因为dfs不需要额外的数据结构，BFS还要使用队列

int ans = 0;

function<bool(int, int, int)> dfs = [&](int cur, int parent, int k) {

// 找到了一条路

if (cur == q) {

ans = k;

return true;

}

for (const int nxt : g\_[cur]) {

if (nxt == parent) continue;

if (dfs(nxt, cur, k + 1)) return true;

}

return false;

};

dfs(p, p, 0);

return ans;

}

private:

unordered\_map<int, vector<int>> g\_;

void buildGraph(TreeNode\* root, TreeNode\* parent) {

if (parent) {

g\_[parent->val].emplace\_back(root->val);

g\_[root->val].emplace\_back(parent->val);

}

if (root->left) buildGraph(root->left, root);

if (root->right) buildGraph(root->right, root);

}

void printGraph() {

for (auto&& [u, neighbours] : g\_) {

printf("%d: [", u);

for (const int v : neighbours) printf("%d ", v);

printf("]\n");

}

}

};

思路和心得：

1、能模块化的就模块化

其实是2道题拼起来的

（1）求LCA Least Common Ancestor最近公共祖先结点，且是2个点一定在树中的情况

（2）求指点结点的深度

2、也可以建图，BFS

但可能不是这道题的本意

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

int findDistance(TreeNode\* root, int p, int q)

{

TreeNode\* LCA = find\_LCA(root, p, q); //先找LCA Least Common Ancestor

int p\_dep = dfs\_calc\_depth(LCA, p);

int q\_dep = dfs\_calc\_depth(LCA, q);

return p\_dep + q\_dep;

}

TreeNode\* find\_LCA(TreeNode\* root, int p, int q)

{

if (root==NULL || root->val == p || root->val == q)

return root;

TreeNode\* L = find\_LCA(root->left, p, q);

TreeNode\* R = find\_LCA(root->right, p, q);

if (L && R)

return root;

else if(L && R==NULL)

return L;

else if (L==NULL && R)

return R;

else

return NULL;

}

int dfs\_calc\_depth(TreeNode\* root, int target)

{

if (root == NULL)

return -1;

if (root->val == target)

return 0;

int L = dfs\_calc\_depth(root->left, target);

int R = dfs\_calc\_depth(root->right, target);

if (L == -1 && R == -1)

return -1;

return max(L , R) + 1;

}

};