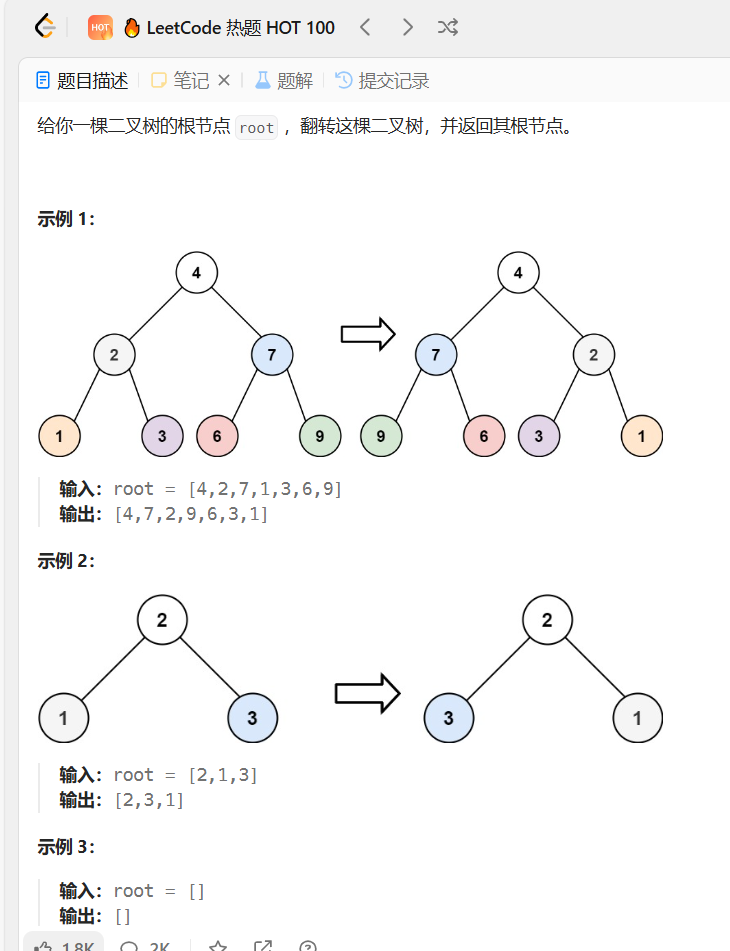
题目：



解题思路一：

递归:

每次进入函数时，形参改为root的左右子树，递归交换即可

#include<iostream>

#include<stack>

using namespace std;

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) {}

};

//递归

class Solution {

public:

TreeNode\* invertTree(TreeNode\* root) {

if (root == nullptr) {

return nullptr;

}

//递归交换左右子树

TreeNode\* left = invertTree(root->left);

TreeNode\* right = invertTree(root->right);

root->left = right;

root->right = left;

return root;

}

};

解题思路二：

辅助栈：

从栈顶元素开始入栈，结束条件是栈空，对于每个节点，先弹出当前栈顶，如果它的左右孩子节点选择不为空的入栈，交换它的左右孩子，然后当前栈顶元素变成下一个循环处理的对象，直到栈为空，代表所有元素都处理完毕

代码：

class Solution {

public:

TreeNode\* invertTree(TreeNode\* root) {

if (root == nullptr) return nullptr;

stack<TreeNode\*> stack;

stack.push(root);//根节点压入栈顶

//从左子树开始深度优先遍历交换左右孩子

while (!stack.empty())

{

TreeNode\* node = stack.top();

stack.pop();

if (node->left != nullptr) stack.push(node->left);

if (node->right != nullptr) stack.push(node->right);

//交换当前节点的左右孩子

TreeNode\* tmp = node->left;

node->left = node->right;

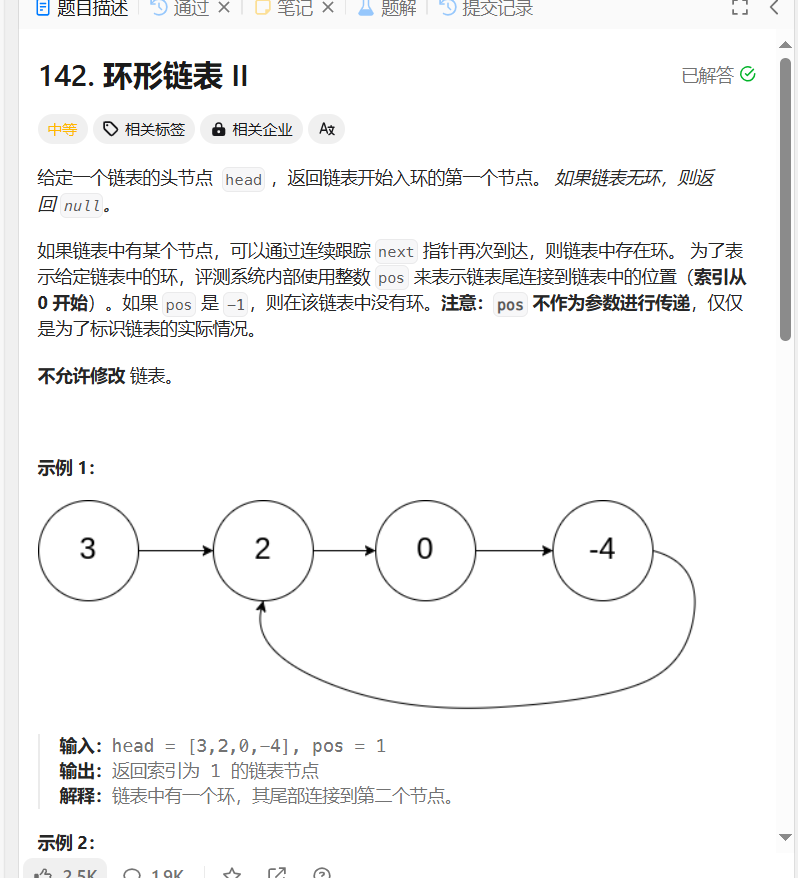
node->right = tmp;

}

return root;

}

};

解题思路一：

快慢指针：

初始化快慢指针等于head，快指针走的速度是慢指针的两倍，如果他俩能够相遇，说明存在环，这时它们第一次相遇，但是我们并不知道相遇的节点是在环的入口节点还是其他节点。由于速度关系，快指针走了2n个环，慢指针走了n个环，假设环的长度是b,头节点到入口的距离是a,那么从头节点出发，走到环的入口节点可能的距离是k=a+nb。这时，我们选择一个指针指向head，假设我们选择把慢指针设置为head,那么这时快指针的走距离是2nb，慢指针走的距离是0，两个指针每次只走一步，在走a步过后，两者相遇，慢指针走了a+0b,快指针走了2nb+a，此时两者相遇的节点就是入口节点，返回两者中任意一个即可

class Solution {

public:

ListNode\* detectCycle(ListNode\* head) {

ListNode\* slow = head;

ListNode\* fast = head;

//非循环链表会有fast为空直接退出

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

//存在环一定会相遇

if (fast == slow) {

slow = head;

//循环结束条件是第二次相遇

while (fast != slow) {

fast = fast->next;

slow = slow->next;

}

return slow;//fast也可以

}

}

return nullptr;//没有环返回空指针

}

};

解题思路二：

哈希表：

使用unordered\_set存储访问过的节点，由于值唯一，每次遍历节点时检查对应的访问节点是否有记录，有记录说明就是环，第一个有记录过的节点就是环的入口节点

class Solution {

public:

ListNode\* detectCycle(ListNode\* head) {

unordered\_set<ListNode\*> visited;

while (head != nullptr) {

if (visited.count(head)) {

return head;

}

visited.insert(head);

head = head->next;

}

return nullptr;

}

};