# 题目

给定一个单链表的头节点head，其中的元素按升序排序，将其转换为平衡二叉搜索树。

示例 1:

输入: head = [-10,-3,0,5,9]

输出: [0,-3,9,-10,null,5]

解释: 一个可能的答案是[0，-3,9，-10,null,5]，它表示所示的高度平衡的二叉搜索树。

示例 2:

输入: head = []

输出: []

提示:

head 中的节点数在[0, 2 \* 104] 范围内

-105 <= Node.val <= 105

# 分析

## 方法一：分治

思路：

我们可以直接模拟「前言」部分的构造方案。

具体地，设当前链表的左端点为left，右端点right，包含关系为「左闭右开」，即left包含在链表中而right不包含在链表中。我们希望快速地找出链表的中位数节点mid。

为什么要设定「左闭右开」的关系？由于题目中给定的链表为单向链表，访问后继元素十分容易，但无法直接访问前驱元素。因此在找出链表的中位数节点mid之后，如果设定「左闭右开」的关系，我们就可以直接用 (left,mid)以及 (mid.next,right)来表示左右子树对应的列表了。并且，初始的列表也可以用(head,null)方便地进行表示，其中null表示空节点。

找出链表中位数节点的方法多种多样，其中较为简单的一种是「快慢指针法」。初始时，快指针fast和慢指针slow均指向链表的左端点left。我们将快指针fast向右移动两次的同时，将慢指针slow向右移动一次，直到快指针到达边界（即快指针到达右端点或快指针的下一个节点是右端点）。此时，慢指针对应的元素就是中位数。

在找出了中位数节点之后，我们将其作为当前根节点的元素，并递归地构造其左侧部分的链表对应的左子树，以及右侧部分的链表对应的右子树。

代码：

class Solution {

public:

ListNode \*getMiddle(ListNode \*left, ListNode \*right) {

ListNode \*fast = left;

ListNode \*slow = left;

while (fast != right && fast->next != right) {

fast = fast->next;

fast = fast->next;

slow = slow->next;

}

return slow;

}

TreeNode \*buildTree(ListNode \*left, ListNode \*right) {

if (left == right) return nullptr;

// 处理ListNode链表

ListNode \*mid = getMiddle(left, right);

// 根据middle构造root

TreeNode \*root = new TreeNode(mid->val);

// 构造Tree

root->left = buildTree(left,mid);

root->right = buildTree(mid->next, right);

return root;

}

TreeNode\* sortedListToBST(ListNode\* head) {

return buildTree(head, nullptr);

}

};

## 方法二：深度优先遍历+广度优先遍历

思路：

BFS构建树，DFS填充节点。

代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

/\*\*

\* 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:

void DFS(ListNode \*&node, TreeNode \*root) {

if (root == NULL) return;

DFS(node, root->left);

root->val = node->val;

node = node->next;

DFS(node, root->right);

}

TreeNode\* sortedListToBST(ListNode\* head) {

if (head == NULL) return NULL;

TreeNode \*root = new TreeNode(0);

ListNode \*node = head;

queue<TreeNode\*> que;

que.push(root);

node = node->next;

while (node) {

TreeNode \*tmpNode = que.front();

que.pop();

tmpNode->left = new TreeNode(0);

que.push(tmpNode->left);

node = node->next;

if (node == nullptr) break;

tmpNode->right = new TreeNode(0);

que.push(tmpNode->right);

node = node->next;

}

DFS(head, root);

return root;

}

};