# 题目

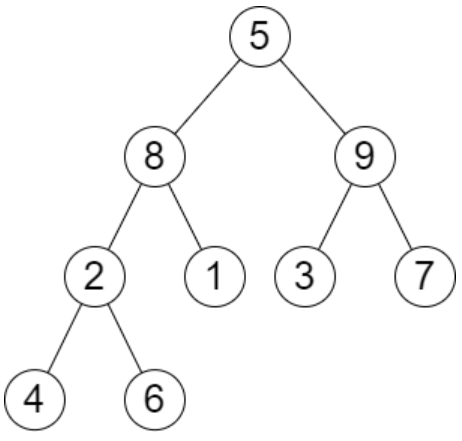
给你一棵二叉树的根节点root和一个正整数k。

树中的层和是指同一层上节点值的总和。

返回树中第k大的层和（不一定不同）。如果树少于k层，则返回-1。

注意，如果两个节点与根节点的距离相同，则认为它们在同一层。

示例 1：



输入：root = [5,8,9,2,1,3,7,4,6], k = 2

输出：13

解释：树中每一层的层和分别是：

- Level 1: 5

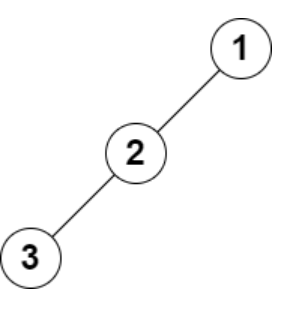
- Level 2: 8 + 9 = 17

- Level 3: 2 + 1 + 3 + 7 = 13

- Level 4: 4 + 6 = 10

第 2 大的层和等于13。

示例 2：



输入：root = [1,2,null,3], k = 1

输出：3

解释：最大的层和是3。

提示：

树中的节点数为 n

2 <= n <= 105

1 <= Node.val <= 106

1 <= k <= n

# 分析

## 方法一：广度优先搜索 + 排序

思路：

先使用广度优先搜索计算出树的每一层的节点值的和，保存在数组levelSumArray中。然后将数组进行排序，返回第k大的值。需要考虑数组长度小于k的边界情况。也可以使用快速选择的算法快速定位第k大的元素。

代码：

class Solution {

public:

long long kthLargestLevelSum(TreeNode\* root, int k) {

queue<TreeNode \*> que;

que.push(root);

vector<long long> levelSumArray;

while (!que.empty()) {

// 层次遍历

long long levelSum = 0, size = que.size();

for (int i = 0; i < size; i++) {

TreeNode \*tmpNode = que.front();

levelSum += tmpNode->val;

que.pop();

if (tmpNode->left) {

que.push(tmpNode->left);

}

if (tmpNode->right) {

que.push(tmpNode->right);

}

}

levelSumArray.push\_back(levelSum);

}

if (levelSumArray.size() < k) return -1;

sort(levelSumArray.rbegin(), levelSumArray.rend());

return levelSumArray[k-1];

}

};

说明：

典型错误为排序使用sort，如下：

sort(levelSumArray.begin(), levelSumArray.end());

return levelSumArray[k-1];

问题出在你使用 sort(levelSumArray.begin(), levelSumArray.end()) 对层和数组进行排序时，它会按照从小到大的顺序排序。但是题目要求返回第 k 大的层和，因此应该将数组按照从大到小排序，然后返回第 k 个元素。