<https://www.lintcode.com/problem/1098/description>

If the depth of a tree is smaller than 5, then this tree can be represented by a list of three-digits integers.

For each integer in this list:

1.The hundreds digit represents the depth D of this node, 1 <= D <= 4.2.The tens digit represents the position P of this node in the level it belongs to, 1 <= P <= 8. The position is the same as that in a full binary tree.3.The units digit represents the value V of this node, 0 <= V <= 9.

Given a list of ascending three-digits integers representing a binary tree with the depth smaller than 5, you need to return the sum of all paths from the root towards the leaves.

**Example 1:**

Input: [113, 215, 221]

Output: 12

Explanation:

The tree that the list represents is:

3

/ \

5 1

The path sum is (3 + 5) + (3 + 1) = 12.

**Example 2:**

Input: [113, 221]

Output: 4

Explanation:

The tree that the list represents is:

3

\

1

The path sum is (3 + 1) = 4.

**Attempt 1: 2022-11-07**

**Solution 1: Preorder recursive traversal + HashMap (60min)**

public class Solution {

/\*\*

\* @param nums: the list

\* @return: the sum of all paths from the root towards the leaves

\*/

int totalSum;

public int pathSumIV(int[] nums) {

totalSum = 0;

Map<Integer, Integer> map = new HashMap<Integer, Integer>();

for(int num : nums) {

int key = num / 10;

int val = num % 10;

map.put(key, val);

}

helper(map, nums[0] / 10, 0);

return totalSum;

}

private void helper(Map<Integer, Integer> map, int key, int preSum) {

int level = key / 10;

int pos = key % 10;

int left = (level + 1) \* 10 + pos \* 2 - 1;

int right = (level + 1) \* 10 + pos \* 2;

int curSum = preSum + map.get(key);

if(!map.containsKey(left) && !map.containsKey(right)) {

totalSum += curSum;

}

if(map.containsKey(left)) {

helper(map, left, curSum);

}

if(map.containsKey(right)) {

helper(map, right, curSum);

}

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(n)

**Refer to**

<https://www.cnblogs.com/grandyang/p/7570954.html>

这道题还是让我们求二叉树的路径之和，但是跟之前不同的是，树的存储方式比较特别，并没有专门的数结点，而是使用一个三位数字来存的，百位数是该结点的深度，十位上是该结点在某一层中的位置，个位数是该结点的结点值。为了求路径之和，我们肯定还是需要遍历树，但是由于没有树结点，所以我们可以用其他的数据结构代替。比如我们可以将每个结点的位置信息和结点值分离开，然后建立两者之间的映射。比如我们可以将百位数和十位数当作key，将个位数当作value，建立映射。由于题目中说了数组是有序的，所以首元素就是根结点，然后我们进行先序遍历即可。在递归函数中，我们先将深度和位置拆分出来，然后算出左右子结点的深度和位置的两位数，我们还要维护一个变量cur，用来保存当前路径之和。如果当前结点的左右子结点不存在，说明此时cur已经是一条完整的路径之和了，加到结果res中，直接返回。否则就是对存在的左右子结点调用递归函数即可

**Solution 2:  Level order iterative traversal  + HashMap (60min)**

public class Solution {

/\*\*

\* @param nums: the list

\* @return: the sum of all paths from the root towards the leaves

\*/

public int pathSumIV(int[] nums) {

Map<Integer, Integer> map = new HashMap<Integer, Integer>();

for(int num : nums) {

int key = num / 10;

int val = num % 10;

map.put(key, val);

}

int totalSum = 0;

Queue<Integer> q = new LinkedList<Integer>();

q.offer(nums[0] / 10);

while(!q.isEmpty()) {

int key = q.poll();

int level = key / 10;

int pos = key % 10;

int left = (level + 1) \* 10 + pos \* 2 - 1;

int right = (level + 1) \* 10 + pos \* 2;

// Find a leaf node, valid path(s) get, add to the result

if(!map.containsKey(left) && !map.containsKey(right)) {

totalSum += map.get(key);

}

// Update left node's value by adding its parent root value

if(map.containsKey(left)) {

q.offer(left);

map.put(left, map.get(left) + map.get(key));

}

// Update right node's value by adding its parent root value

if(map.containsKey(right)) {

q.offer(right);

map.put(right, map.get(right) + map.get(key));

}

}

return totalSum;

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(n)

**Refer to**

<https://www.cnblogs.com/grandyang/p/7570954.html>

下面这种方法是迭代的形式，我们使用的层序遍历，与先序遍历不同的是，我们不能维护一个当前路径之和的变量，这样会重复计算结点值，而是在遍历每一层的结点时，加上其父结点的值，如果某一个结点没有子结点了，才将累加起来的结点值加到结果res中