# 题目

对于一棵深度小于 5 的树，可以用一组三位十进制整数来表示。给定一个由三位数组成的 递增 的数组 nums 表示一棵深度小于 5 的二叉树，对于每个整数：

- 百位上的数字表示这个节点的深度d，1 <= d <= 4。

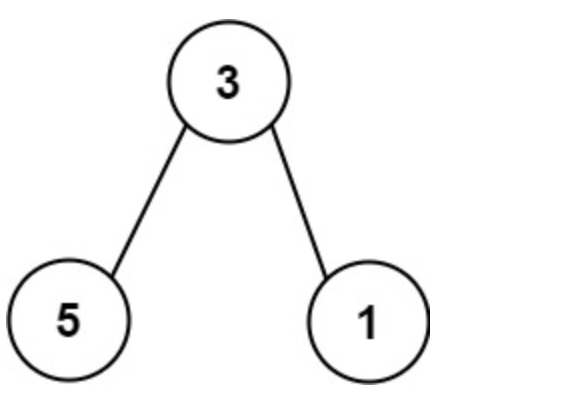
- 十位上的数字表示这个节点在当前层所在的位置p，1 <= p <= 8。位置编号与一棵满二叉树的位置编号相同。

- 个位上的数字表示这个节点的权值v，0 <= v <= 9。

返回从根到所有叶子结点的路径之和。

保证给定的数组表示一个有效的连接二叉树。

示例 1：



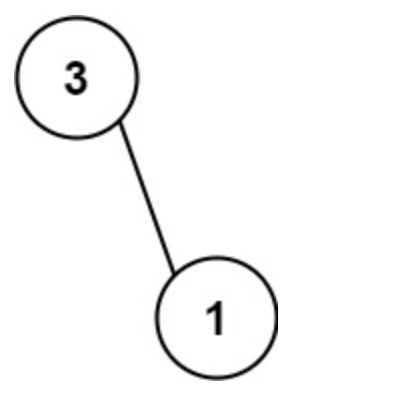
输入: nums = [113, 215, 221]

输出: 12

解释: 列表所表示的树如上所示。

路径和 = (3 + 5) + (3 + 1) = 12。

示例 2：



输入: nums = [113, 221]

输出: 4

解释: 列表所表示的树如上所示。

路径和 = (3 + 1) = 4。

提示:

1 <= nums.length <= 15

110 <= nums[i] <= 489

nums 表示深度小于 5 的有效二叉树

nums 以升序排序。

# 分析

要解决这个问题，我们需要先根据“三位十进制整数”的规则构建二叉树结构，再遍历所有从根到叶子的路径，计算路径上节点值的总和。核心是将数字编码映射为树的层级和位置关系，并通过深度优先搜索（DFS）遍历路径。

思路分析：

1、数字编码解析：每个三位数 abc 对应节点的属性：

- 百位 a：深度 d（1≤d≤4，根为深度1）。

- 十位 b：当前层的位置 p（1≤p≤8，按满二叉树的位置编号，左子节点位置为 2p-1，右子节点为 2p）。

- 个位 c：节点的权值 v。

2、构建树的映射表：用哈希表（或数组）存储“深度-位置”到“节点值”的映射，便于快速查询任意节点的父/子节点是否存在及对应的值。

- 键：d \* 10 + p（将深度和位置合并为一个两位数，如深度2、位置1对应键 21）。

- 值：节点的权值 v。

3、深度优先搜索（DFS）计算路径和：

- 从根节点（深度1、位置1）开始，递归遍历所有子节点。

- 若当前节点无左、右子节点（叶子节点），将当前路径的和加入总结果。

- 若有子节点，递归遍历子节点并累加路径值。

代码：

class Solution {

private:

int total = 0; // 存储所有路径和的总和

unordered\_map<int, int> nodeMap; // 键：d\*10+p，值：节点权值v

// DFS函数：当前深度d，当前位置p，当前路径的累加和currSum

void dfs(int d, int p, int currSum) {

// 计算当前节点的键，获取节点值

int key = d \* 10 + p;

int val = nodeMap[key];

currSum += val; // 更新当前路径和

// 计算当前节点的左、右子节点的位置（满二叉树规则）

int leftP = 2 \* p - 1; // 左子节点位置

int rightP = 2 \* p; // 右子节点位置

int nextD = d + 1; // 子节点深度（当前深度+1）

// 检查左、右子节点是否存在（通过子节点的键是否在map中）

bool hasLeft = nodeMap.count(nextD \* 10 + leftP);

bool hasRight = nodeMap.count(nextD \* 10 + rightP);

// 情况1：当前节点是叶子节点（无左、右子节点），将路径和加入总结果

if (!hasLeft && !hasRight) {

total += currSum;

return;

}

// 情况2：有左子节点，递归遍历左子树

if (hasLeft) {

dfs(nextD, leftP, currSum);

}

// 情况3：有右子节点，递归遍历右子树

if (hasRight) {

dfs(nextD, rightP, currSum);

}

}

public:

int pathSum(vector<int>& nums) {

// 步骤1：构建节点映射表

for (int num : nums) {

int d = num / 100; // 提取深度（百位）

int p = (num / 10) % 10; // 提取位置（十位）

int v = num % 10; // 提取权值（个位）

nodeMap[d \* 10 + p] = v; // 存入map

}

// 步骤2：从根节点（d=1, p=1）开始DFS，初始路径和为0

dfs(1, 1, 0);

return total;

}

};

代码解释：

1、构建映射表：

- 遍历输入数组nums，对每个数字分解出深度d、位置p和权值v，并将d\*10+p作为键、v作为值存入nodeMap。

- 例如，数字215分解为d=2、p=1、v=5，对应键21，值5。

2、DFS遍历路径：

- 初始调用 dfs(1, 1, 0)，从根节点（深度1、位置1）开始，初始路径和为0。

- 每次递归先计算当前节点的键和值，将值加入当前路径和 currSum。

- 根据满二叉树规则，计算左子节点（2p-1）和右子节点（2p）的位置，检查子节点是否存在（通过 nodeMap的 count方法）。

- 若为叶子节点（无左右子节点），将当前路径和加入 total；否则递归遍历存在的子节点。

3、返回结果：所有路径遍历完成后，total即为根到所有叶子的路径和，返回total。

复杂度分析

- 时间复杂度：O(n)，其中n是nums的长度（即树的节点数）。

构建映射表遍历n个节点，时间O(n)。

DFS 遍历每个节点恰好一次，时间O(n)。

整体为线性时间，效率极高。

- 空间复杂度：O(n)。

映射表nodeMap存储n个节点，空间O(n)。

DFS 递归栈的深度等于树的深度（最大为4，因树深度小于5），空间可视为 `O(1)`。

总空间由映射表主导，为 O(n)。

该方法通过映射表快速解析节点关系，结合DFS高效遍历所有路径，完美适配题目约束（树深度≤4，节点数≤15），逻辑清晰且易于维护。