Path Sum IV

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`
2. The tens digit represents the position `P` of this node in the level it belongs to, `1 . The position is the same as that in a full binary tree.`
3. The units digit represents the value `V` of this node, `0`

Given a list of `ascending` three-digits integers representing a binary 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.
```

Solution 1

How do we solve problem like this if we were given a normal tree? Yes, traverse it, keep a root to leaf running sum. If we see a leaf node (node.left == null && node.right == null), we add the running sum to the final result.

Now each tree node is represented by a number. 1st digits is the `level`, 2nd is the `position` in that `level` (note that it starts from `1` instead of `0`). 3rd digit is the value. We need to find a way to traverse this `tree` and get the sum.

The idea is, we can form a `tree` using a HashMap. The `key` is first two digits which marks the position of a node in the tree. The `value` is value of that node. Thus, we can easily find a node's left and right children using math.
Formula: For node `xy?` its left child is `(x+1)(y*2-1)?` and right child is `(x+1)(y*2)?`

Given above HashMap and formula, we can traverse the `tree`. Problem is solved!

```
class Solution {
    int sum = 0;
    Map<Integer, Integer> tree = new HashMap<>();

    public int pathSum(int[] nums) {
        if (nums == null || nums.length == 0) return 0;

        for (int num : nums) {
            int key = num / 10;
            int value = num % 10;
            tree.put(key, value);
        }

        traverse(nums[0] / 10, 0);

        return sum;
    }

    private void traverse(int root, int preSum) {
        int level = root / 10;
        int pos = root % 10;
        int left = (level + 1) * 10 + pos * 2 - 1;
        int right = (level + 1) * 10 + pos * 2;

        int curSum = preSum + tree.get(root);

        if (!tree.containsKey(left) && !tree.containsKey(right)) {
            sum += curSum;
            return;
        }

        if (tree.containsKey(left)) traverse(left, curSum);
        if (tree.containsKey(right)) traverse(right, curSum);
    }
}
```

written by shawngao original link here

Solution 2

```
          0
      0       1
   0   1     2   3
0 1  2 3   4 5  6 7
```

Regardless whether these nodes exist:

the position of left child is always `parent_pos * 2`;
the position of right child is always `parent_pos * 2 + 1`;
the position of parent is always `child_pos / 2`;

Solution C++ Array

```
class Solution {
public:
    int pathSum(vector<int>& nums) {
        int m[5][8] = {};
        for (int n : nums) {
            int i = n / 100; // i is 1 based index;
            int j = (n % 100) / 10 - 1; // j used 0 based index;
            int v = n % 10;
            m[i][j] = m[i - 1][j / 2] + v;
        }

        int sum = 0;
        for (int i = 1; i < 5; i++) {
            for (int j = 0; j < 8; j++) {
                if (i == 4 || m[i][j] && !m[i + 1][j * 2] && !m[i + 1][j * 2 + 1]){
                    sum += m[i][j];
                }
            }
        }
        return sum;
    }
};
```

Solution C++ map

If we use map, we don't need to do the boundary check at little extra cost of memory.

```
class Solution {
public:
    int pathSum(vector<int>& nums) {
        map<int, map<int, int>> m;
        for (int n : nums) {
            int i = n / 100 - 1; // i is 0 based index;
            int j = (n % 100) / 10 - 1; // j used 0 based index;
            int v = n % 10;
            m[i][j] = m[i - 1][j / 2] + v;
        }

        int sum = 0;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 8; j++) {
                sum += m[i][j] && !m[i + 1][j * 2] && !m[i + 1][j * 2 + 1] ? m[i][j
] : 0;
            }
        }
        return sum;
    }
};
```

Solution C++ - queue

```
class Solution {
public:
    int pathSum(vector<int>& nums) {
        if (nums.empty()) return 0;
        int sum = 0;
        queue<info> q;
        info dummy(0);
        info* p = &dummy; // parent start with dummy info, root have no real parent
;
        for (int n : nums) {
            info c(n); // child;
            while (!p->isparent(c) && !q.empty()) {
                sum += p->leaf ? p->v : 0;
                p = &q.front();
                q.pop();
            }
            p->leaf = false;
            c.v += p->v;
            q.push(c);
        }
        while (!q.empty()) {
            sum += q.front().v;
            q.pop();
        }
        return sum;
    }
private:
    struct info {
        int i, j, v;
        bool leaf;
        info(int n) : i(n / 100 - 1), j((n % 100) / 10 - 1), v(n % 10), leaf(true)
{};
        bool isparent(info other) { return i == other.i - 1 && j == other.j / 2;};
    };
};
```

Solution Java

```java
class Solution {
    public int pathSum(int[] nums) {
        int[][] m = new int[5][8];
        for (int n : nums) {
            int i = n / 100; // i is 1 based index;
            int j = (n % 100) / 10 - 1; // j used 0 based index;
            int v = n % 10;
            m[i][j] = m[i - 1][j / 2] + v;
        }

        int sum = 0;
        for (int i = 1; i < 5; i++) {
            for (int j = 0; j < 8; j++) {
                if (i == 4 || m[i][j] != 0 && m[i + 1][j * 2] == 0 && m[i + 1][j *
2 + 1] == 0){
                    sum += m[i][j];
                }
            }
        }
        return sum;
    }
}
```

written by alexander original link here

Solution 3

The idea is to first represent the tree as a map, that contains the tree in the form of Node position number --> Node value , where the Node number is the position of the node in a complete tree.

Then, we just do a dfs on the above tree to accumulate all the root to leaf path sums.

```
Lets take a tree example
                                            10
                                           /  \
                                         20      30
                                        / \      / \
                                      40  50  60   70
```

For the above tree,
the node positions are as follows,
root's position = 1,
left child position = parent_position * 2
right child position = parent_position * 2 + 1
Based on the above complete tree positions, the map generated for the above example would be
1 --> 10
2 --> 20
3 --> 30
4 --> 40
5 --> 50
6 --> 60
7 --> 70

To generate these positions from the given input,
if the input number is 314
we extract the digits to level = 3, positionInLevel = 1, value = 4
and the formula I arrived at to get the node's position in complete tree = [2 ^ (level-1)] + positionInLevel - 1

Now that we have the above map generated,
we do a simple dfs starting from the root, and keep accumulating the sum, once we reach a leaf, we add the accumulated sum to the result.

```java
    public int pathSum(int[] nums) {
        Map<Integer, Integer> positionToNodeMap = new HashMap<>();
        Arrays.stream(nums).forEach( num -> {
            int[] digits = IntStream.range(0, 3).map(i -> (num + "").charAt(i) - '0').toArray();
            positionToNodeMap.put((int)Math.pow(2, digits[0] - 1) - 1 + digits[1], digits[2]);
        });
        int[] res = new int[1];
        dfs(1, 0, res, positionToNodeMap);
        return res[0];
    }

   private void dfs(int cur, int sum, int[] res, Map<Integer, Integer> map) {
        if(!map.containsKey(cur)) return;
        int left = cur * 2, right = cur * 2 + 1, totalSum = sum + map.get(cur);
        if(!map.containsKey(left) && !map.containsKey(right)) { res[0] += totalSum; return; } // Leaf node
        dfs(left, totalSum, res, map);
        dfs(right, totalSum, res, map);
    }
```

written by johnyrufus16 original link here