Problem 666: Path Sum IV

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

If the depth of a tree is smaller than

5

, then this tree can be represented by an array of three-digit integers. You are given an

ascending

array

nums

consisting of three-digit integers representing a binary tree with a depth smaller than

5

, where for each integer:

The hundreds digit represents the depth

d

of this node, where

1 <= d <= 4

.

The tens digit represents the position

p

of this node within its level, where

1 <= p <= 8

, corresponding to its position in a

full binary tree

.

The units digit represents the value

v

of this node, where

0 <= v <= 9

.

Return the

sum

of

all paths

from the

root

towards the

leaves

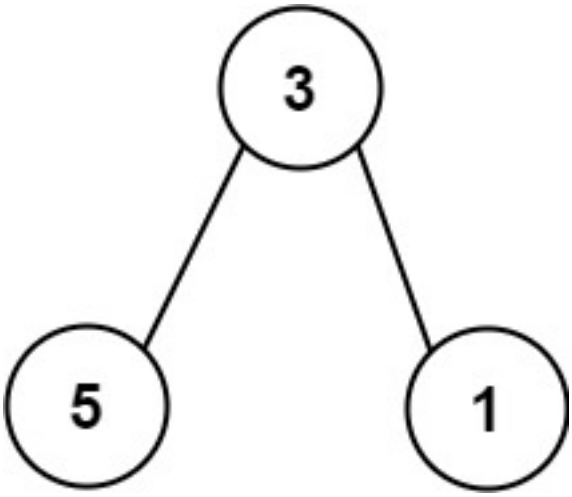
.

It is

guaranteed

that the given array represents a valid connected binary tree.

Example 1:



Input:

nums = [113,215,221]

Output:

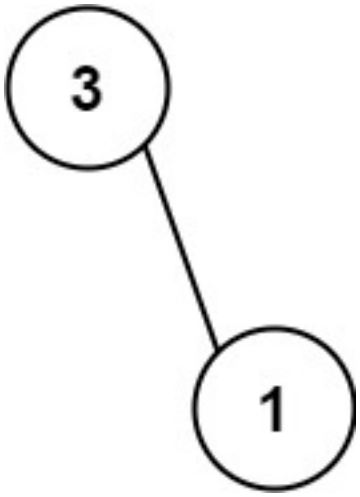
12

Explanation:

The tree that the list represents is shown.

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

Example 2:



Input:

nums = [113,221]

Output:

4

Explanation:

The tree that the list represents is shown.

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

Constraints:

1 <= nums.length <= 15

110 <= nums[i] <= 489

nums

represents a valid binary tree with depth less than

5

.

nums

is sorted in ascending order.

Code Snippets

C++:

```
class Solution {
public:
int pathSum(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int pathSum(int[] nums) {

}
}
```

Python3:

```
class Solution:
def pathSum(self, nums: List[int]) -> int:
```

Python:

```
class Solution(object):
def pathSum(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
```

```
 * @return {number}
 */
var pathSum = function(nums) {

};
```

TypeScript:

```
function pathSum(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int PathSum(int[] nums) {

}
}
```

C:

```
int pathSum(int* nums, int numsSize) {

}
```

Go:

```
func pathSum(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun pathSum(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func pathSum(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn path_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def path_sum(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function pathSum($nums) {

}
}
```

Dart:

```
class Solution {
int pathSum(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def pathSum(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec path_sum(nums :: [integer]) :: integer
def path_sum(nums) do

end
end
```

Erlang:

```
-spec path_sum(Nums :: [integer()]) -> integer().
path_sum(Nums) ->
.
```

Racket:

```
(define/contract (path-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Path Sum IV
* Difficulty: Medium
* Tags: array, tree, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
class Solution {
public:
int pathSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Path Sum IV
* Difficulty: Medium
* Tags: array, tree, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int pathSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Path Sum IV
Difficulty: Medium
Tags: array, tree, hash, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def pathSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def pathSum(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Path Sum IV
* Difficulty: Medium
* Tags: array, tree, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} nums
* @return {number}
*/
var pathSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Path Sum IV
* Difficulty: Medium
* Tags: array, tree, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

function pathSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Path Sum IV
* Difficulty: Medium
* Tags: array, tree, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int PathSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Path Sum IV
* Difficulty: Medium
* Tags: array, tree, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int pathSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Path Sum IV
// Difficulty: Medium
// Tags: array, tree, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func pathSum(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun pathSum(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func pathSum(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Path Sum IV
// Difficulty: Medium
// Tags: array, tree, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn path_sum(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def path_sum(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function pathSum($nums) {

}
}
```

Dart Solution:

```
class Solution {
int pathSum(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def pathSum(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec path_sum(nums :: [integer]) :: integer
def path_sum(nums) do

end
end
```

Erlang Solution:

```
-spec path_sum(Nums :: [integer()]) -> integer().
path_sum(Nums) ->
.
```

Racket Solution:

```
(define/contract (path-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```