Problem 1104: Path In Zigzag Labelled Binary Tree

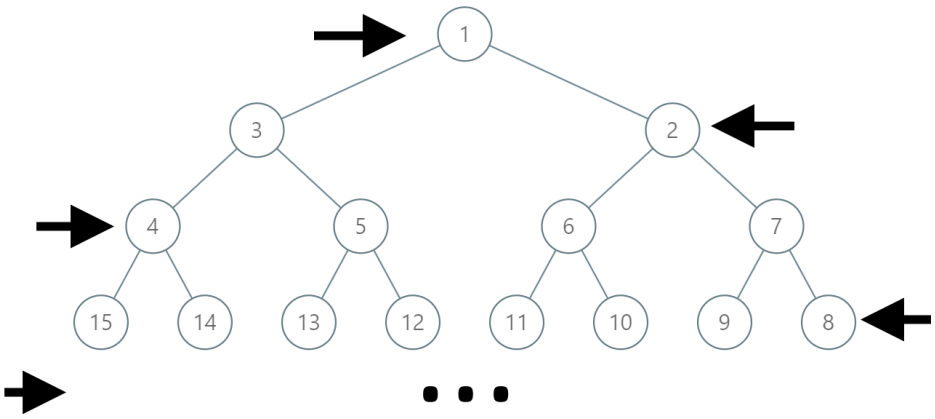
Problem Information

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

Problem Description

In an infinite binary tree where every node has two children, the nodes are labelled in row order.

In the odd numbered rows (ie., the first, third, fifth,...), the labelling is left to right, while in the even numbered rows (second, fourth, sixth,...), the labelling is right to left.



Given the

label

of a node in this tree, return the labels in the path from the root of the tree to the node with that

label

.

Example 1:

Input:

label = 14

Output:

[1,3,4,14]

Example 2:

Input:

label = 26

Output:

[1,2,6,10,26]

Constraints:

1 <= label <= 10^6

Code Snippets

C++:

```
class Solution {
public:
vector<int> pathInZigZagTree(int label) {

}
};
```

Java:

```
class Solution {
public List<Integer> pathInZigZagTree(int label) {

}
}
```

Python3:

```
class Solution:
def pathInZigZagTree(self, label: int) -> List[int]:
```

Python:

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

JavaScript:

```
/**
* @param {number} label
* @return {number[]}
*/
var pathInZigZagTree = function(label) {

};
```

TypeScript:

```
function pathInZigZagTree(label: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> PathInZigZagTree(int label) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* pathInZigZagTree(int label, int* returnSize) {

}
```

Go:

```
func pathInZigZagTree(label int) []int {

}
```

Kotlin:

```
class Solution {
fun pathInZigZagTree(label: Int): List<Int> {

}
}
```

Swift:

```
class Solution {
func pathInZigZagTree(_ label: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn path_in_zig_zag_tree(label: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} label
# @return {Integer[]}
```

```
def path_in_zig_zag_tree(label)

end
```

PHP:

```
class Solution {

/**
* @param Integer $label
* @return Integer[]
*/
function pathInZigZagTree($label) {

}
}
```

Dart:

```
class Solution {
List<int> pathInZigZagTree(int label) {

}
}
```

Scala:

```
object Solution {
def pathInZigZagTree(label: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec path_in_zig_zag_tree(label :: integer) :: [integer]
def path_in_zig_zag_tree(label) do

end
end
```

Erlang:

```
-spec path_in_zig_zag_tree(Label :: integer()) -> [integer()].
path_in_zig_zag_tree(Label) ->
.
```

Racket:

```
(define/contract (path-in-zig-zag-tree label)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Path In Zigzag Labelled Binary Tree
* Difficulty: Medium
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> pathInZigZagTree(int label) {

}
};
```

Java Solution:

```
/**
* Problem: Path In Zigzag Labelled Binary Tree
* Difficulty: Medium
* Tags: tree, math
*
* Approach: DFS or BFS traversal
```

```
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public List<Integer> pathInZigZagTree(int label) {

}
}
```

Python3 Solution:

```
"""
Problem: Path In Zigzag Labelled Binary Tree
Difficulty: Medium
Tags: tree, math

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Path In Zigzag Labelled Binary Tree
* Difficulty: Medium
```

```
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number} label
* @return {number[]}
*/
var pathInZigZagTree = function(label) {

};
```

TypeScript Solution:

```
/**
* Problem: Path In Zigzag Labelled Binary Tree
* Difficulty: Medium
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

function pathInZigZagTree(label: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Path In Zigzag Labelled Binary Tree
* Difficulty: Medium
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

public class Solution {
public IList<int> PathInZigZagTree(int label) {

}
}
```

C Solution:

```
/*
* Problem: Path In Zigzag Labelled Binary Tree
* Difficulty: Medium
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* pathInZigZagTree(int label, int* returnSize) {

}
```

Go Solution:

```
// Problem: Path In Zigzag Labelled Binary Tree
// Difficulty: Medium
// Tags: tree, math
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

func pathInZigZagTree(label int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun pathInZigZagTree(label: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func pathInZigZagTree(_ label: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Path In Zigzag Labelled Binary Tree
// Difficulty: Medium
// Tags: tree, math
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn path_in_zig_zag_tree(label: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} label
# @return {Integer[]}
def path_in_zig_zag_tree(label)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $label
* @return Integer[]
*/
function pathInZigZagTree($label) {

}
}
```

Dart Solution:

```
class Solution {
List<int> pathInZigZagTree(int label) {

}
}
```

Scala Solution:

```
object Solution {
def pathInZigZagTree(label: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec path_in_zig_zag_tree(label :: integer) :: [integer]
def path_in_zig_zag_tree(label) do

end
end
```

Erlang Solution:

```
-spec path_in_zig_zag_tree(Label :: integer()) -> [integer()].
path_in_zig_zag_tree(Label) ->
.
```

Racket Solution:

```
(define/contract (path-in-zig-zag-tree label)
(-> exact-integer? (listof exact-integer?))
)
```