1104. Path In Zigzag Labelled Binary Tree

Binary Tree

Problem Description

Math

In this problem, we are given a representation of an infinite binary tree where the nodes are labeled in row order. However, the labeling pattern alternates between each row: in the odd-numbered rows (first, third, fifth, etc.), the nodes are labeled from left to right, while in the even-numbered rows (second, fourth, sixth, etc.), the labels are ordered from right to left. This binary tree forms what is known as a "zigzag" pattern. The task is to find the path from the root of this binary tree to a given node, identified by its label, and return the labels of all the nodes along this path.

Intuition

To solve this problem, the key observation is that the labeling sequence is straightforward to generate for the first row but

reverses at every new row level. So if we can determine which level the label falls on, we could backtrack from the label to the tree's root by inverting the labeling at each step.

Medium

First, we need to identify the row in which the target label exists. We do this by testing where label fits within the doubling sequence of 1, 2, 4, 8, 16, etc., which represents the increasing maximum label number at each row of the binary tree. Once we know the row, we can find each ancestor's label by calculating the parent label, which is half the label value of the current node if the labels were in a perfect binary tree without zigzag. However, due to the zigzag labeling, the actual label must

be found by reflecting the perfect binary tree parent label over the middle of the range of possible labels for that level. The solution follows these steps:

Iteratively find each label on the path from the node label up to the root by reflecting the theoretical perfect tree labels. This

involves computing the range of possible labels for the current level and finding the reflection of label along the middle of

Determine the level i of the label by finding the highest power of 2 less than or equal to the label. Initialize an array ans to have a size equal to the level i (since the path will contain i elements from the root to the label).

Repeat step 3 until we reach the tree's root (the topmost level).

this range. After reflecting, we divide the label by 2 to move up to the parent level in the next iteration.

- The implementation of the solution follows a certain logical approach which can be dissected as follows: The algorithm starts by initializing two variables, x and 1, with the value of 1. The variable x tracks the starting label of the current row in the tree, while i is used to keep track of the current row level we are examining.
- A while loop is then used to find out the level i on which the input label is present. This is done by sequentially doubling x (x <<= 1 is equivalent to $x \neq = 2$) and incrementing i until x becomes larger than label. At this point, x represents the

Decrement the level i as we move up the tree toward the root.

that contains the label.

position in the ans array.

root in reverse.

Example Walkthrough

Solution Approach

path from label to the root.

A while loop is then used to populate the ans array by repeatedly finding the parent label for label. This is the crux of the algorithm:

starting label of the next row (the row after the one containing label), and i is the level number corresponding to the row

Following the identification of the level, an array ans is initialized with a size of i. This array is used to store the labels in the

 Calculate the parent label as if it were a non-zigzag (or perfect binary tree) by dividing the current label by 2. Adjust the parent label to account for the zigzag pattern. This is done by calculating the reflection of the label using the formula ((1 << (i - 1)) + (1 << i) - 1 - label) >> 1. Essentially, it finds the theoretical mirror position of the label in the non-zigzag perfect binary tree within the current row and then divides by 2 to reach the parent label for the next higher row.

Each iteration of the loop calculates the label for the node on the next higher level and assigns this label to the respective

This process is repeated until the root of the tree is reached, effectively constructing the path from the target label to the

Assign label to its corresponding position in the array ans (given that arrays are 0-indexed, the correct position is i - 1).

Once the root is reached (which would happen when i is decremented to 0), the ans array is complete and is returned as the final result representing the path from the root to the input label. Through these steps, we are able to circumvent the more complex task of directly simulating the entire path in a zigzag patterned

tree, and instead, we use mathematical patterns to efficiently compute the parent-child relationship and find the desired path.

Let's consider finding the path to the label 14 in our zigzag labeled binary tree to illustrate the solution approach.

We begin by initializing two variables x and i to 1. x will help us find the start of the row that contains our label 14, and i keeps track of the level of the row.

We use a while loop to identify the level i our label 14 is located at. We start at x = 1 (level 1) and keep doubling x and

increment i until x (which doubles each time, signifying the start of the next row) is greater than 14. After this loop, x

∘ To find the parent label, we normally would divide 14 by 2, which is 7. However, because of the zigzag pattern, 7 isn't the correct label. We

need to find the reflection of label 7 in the non-zigzag tree for level 3. The actual parent label in the zigzag tree is calculated as ((1 << 2)

We then repeat this reflection calculation and populate the ans array until we reach the root. The next label to compute would

be using 5, leading to a parent label of 2. Continuing this, we'd eventually reach the root and have the ans array filled with

This process repeats until the entire path from label 14 back to the root is found. Every iteration calculates the label's parent

equals 16 and i is 4, meaning label 14 is in row 4. We then initialize an array ans to hold the labels in the path from label 14 to the root. The size of this array will be the level i

Now, we populate the ans array with the parent labels in the path from 14 up to the root. This is the trickiest part:

First, we assign the current label 14 to ans[3] (since i is 4 and arrays are 0-indexed).

Determine the level of the tree where the label is. The levels in the tree

Initialize an array to store the path from the root to the label

Set the current position in the path array to the label

Calculate the label's parent in the next higher level.

Zigzag pattern means we have to invert the label within its level

// Initialize the level to 1 and the start value of that level (x) to 1.

// Determine the level of the tree where the label is located.

// Starting from the label's level, move up to the root.

// Add the current label to the path.

// then adjust for the zigzag pattern.

label = (levelStart + levelEnd - label) / 2;

for (int currentLevel = level; currentLevel > 0; --currentLevel) {

// Calculate the parent label in the previous level of a perfect binary tree,

// Since we've built the path from the bottom up, reverse it to get the correct order.

int levelStart = (1 << (currentLevel - 1)); // Start of the current level</pre>

int levelEnd = (1 << currentLevel) - 1; // End of the current level</pre>

Working back up the tree from the label to the root

root to the label 14 in the zigzag patterned binary tree.

def pathInZiqZaqTree(self, label: int) -> List[int]:

level_start_value = level_index = 1

path[level index - 1] = label

public List<Integer> pathInZigZagTree(int label) {

while ((startOfLevel * 2) <= label) {</pre>

path = [0] * level_index

while level index:

return path

Java

class Solution {

+ $(1 \ll 3) - 1 - 7) >> 1$. Computing this gives us 5. So ans [2] is set to 5. We decrement i to move up a level.

we found, which is 4.

Solution Implementation

from typing import List

Python

class Solution:

labels [1, 3, 5, 14].

- in the subsequent upper row and assigns it to the ans array. Once we have the complete path, the ans array [1, 3, 5, 14] is returned as the result, which represents the path from the
- # double in number each time (1, 2, 4, 8, ...), hence the use of bit shifting # to represent this binary progression. The level_index keeps track of the depth. while (level start value << 1) <= label:</pre> level start value <<= 1</pre> level_index += 1
- label = ((1 << (level_index 1)) + (1 << level_index) 1 label) >> 1 level_index -= 1 # Return the path that was constructed

```
++level;
// Create a list to store the path from root to the label.
List<Integer> path = new ArrayList<>();
```

int level = 1;

int startOfLevel = 1;

startOfLevel *= 2;

path.add(label);

Collections.reverse(path);

return path;

C++

#include <vector>

#include <algorithm>

```
class Solution {
public:
    // Method to find the path from the root to a given label in a zigzag labelled binary tree
    vector<int> pathInZigZagTree(int label) {
        // Initialize root level as 1, and depth as 1
        int levelStartValue = 1, depth = 1;
        // Calculate the depth of the given label. The depth increases while it is possible to go further down
        while ((levelStartValue << 1) <= label) {</pre>
            levelStartValue <<= 1;</pre>
            ++depth;
        // Prepare a vector to store the path
        vector<int> path;
        // Loop from the level of the label to the root
        for (; depth > 0; --depth) {
            // Add current label to the path
            path.push back(label);
            // Find the parent label. The operation calculates the opposite label in the same level and then finds the parent
            label = ((1 << (depth - 1)) + (1 << depth) - 1 - label) >> 1;
        // Reverse the path to start from the root
        reverse(path.begin(), path.end());
        // Return the path from the root to the label
        return path;
};
TypeScript
// Import necessary TypeScript feature(s)
import { reverse } from 'lodash';
// Function to find the path from the root to a given label in a zigzag labeled binary tree
function pathInZigZagTree(label: number): number[] {
    // Initialize root level value as 1, and depth as 1
    let levelStartValue: number = 1;
    let depth: number = 1;
    // Calculate the depth of the given label
    // The depth increases while it is possible to go further down
    while ((levelStartValue << 1) <= label) {</pre>
        levelStartValue <<= 1;</pre>
```

// Find the parent label. This operation calculates the opposite label in the same level and then finds

// the parent in the previous level by performing integer division by 2

double in number each time (1, 2, 4, 8, ...), hence the use of bit shifting

Initialize an array to store the path from the root to the label

Set the current position in the path array to the label

Calculate the label's parent in the next higher level.

Zigzag pattern means we have to invert the label within its level

label = ((1 << (level_index - 1)) + (1 << level_index) - 1 - label) >> 1

Working back up the tree from the label to the root

to represent this binary progression. The level_index keeps track of the depth.

label = (((1 << (depth - 1)) + (1 << depth) - 1 - label) >> 1);

```
def pathInZiqZaqTree(self, label: int) -> List[int]:
    level_start_value = level_index = 1
   # Determine the level of the tree where the label is. The levels in the tree
```

class Solution:

depth++;

reverse(path);

return path;

// Example use of the function

const label: number = 14;

from typing import List

// Prepare an array to store the path

// Add current label to the path

// Loop from the level of the label to the root

// Reverse the path so that it starts from the root

// Return the path from the root to the label

const pathFromRoot: number[] = pathInZigZagTree(label);

while (level start value << 1) <= label:</pre>

path[level index - 1] = label

Return the path that was constructed

label, this loop runs in O(log(label)) time.

level start value <<= 1</pre>

level_index += 1

path = [0] * level_index

level_index -= 1

while level index:

return path

console.log(pathFromRoot); // Output the path to the console

const path: number[] = [];

for (; depth > 0; depth--) {

path.push(label):

```
Time and Space Complexity
  The given Python function pathInZigZagTree calculates the path from the root of a zigzag-labelled binary tree to a node labelled
  label. The tree starts with the root labelled 1 and follows a zigzag pattern such that each successive layer reverses the order of
  numbers relative to the previous layer.
  Time Complexity:
```

1. The initial while loop runs as long as 2¹ is less than or equal to label. Since the height of the tree increases logarithmically with respect to the

2. The second while loop constructs the path in reverse, starting from label and moving up the tree one level at a time. As the height of the tree

Thus, the overall time complexity of the function is O(log(label)) + O(log(label)) = O(log(label)).

1. Space is used for the list ans, which has a length equal to the height of the tree. The height of the tree for node label is O(log(label)), hence the space complexity for ans is O(log(label)). 2. A constant amount of auxiliary space is used for variables x and i.

Thus, the overall space complexity of the function is O(log(label)).

for a given label is log(label), this loop will also execute O(log(label)) times.

3. Inside the second while loop, there are only constant time operations.

Space Complexity: