

1104. Path In Zigzag Labelled Binary Tree

MediumTreeMathBinary Tree

Leetcode Link

Problem Description

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.

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:

- 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`).
- 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 this range. After reflecting, we divide the label by 2 to move up to the parent level in the next iteration.
- Repeat step 3 until we reach the tree's root (the topmost level).

Solution Approach

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 `i`, 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 *= 2`) and incrementing `i` until `x` becomes larger than `label`. At this point, `x` represents the starting label of the next row (the row after the one containing `label`), and `i` is the level number corresponding to the row that contains the `label`.
- 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 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:
 - Assign `label` to its corresponding position in the array `ans` (given that arrays are 0-indexed, the correct position is `i - 1`).
 - 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.
 - Decrement the level `i` as we move up the tree toward the root.
- Each iteration of the loop calculates the label for the node on the next higher level and assigns this label to the respective position in the `ans` array.
- This process is repeated until the root of the tree is reached, effectively constructing the path from the target `label` to the root in reverse.
- 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.

Example Walkthrough

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` 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` we found, which is 4.
- 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).
 - 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 $((1 << 2) + (1 << 3) - 1 - 7) >> 1$. Computing this gives us 5. So `ans[2]` is set to 5.
 - We decrement `i` to move up a level.
- We 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 labels [1, 3, 5, 14].
- This process repeats until the entire path from label 14 back to the root is found. Every iteration calculates the label's parent 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 root to the label 14 in the zigzag patterned binary tree.

Python Solution

```
1 from typing import List
2
3 class Solution:
4     def pathInZigZagTree(self, label: int) -> List[int]:
5         level_start_value = level_index = 1
6
7         # Determine the level of the tree where the label is. The levels in the tree
8         # double in number each time (1, 2, 4, 8, ...), hence the use of bit shifting
9         # to represent this binary progression. The level_index keeps track of the depth.
10        while (level_start_value <= 1) <= label:
11            level_start_value <<= 1
12            level_index += 1
13
14        # Initialize an array to store the path from the root to the label
15        path = [0] * level_index
16
17        # Working back up the tree from the label to the root
18        while level_index:
19            # Set the current position in the path array to the label
20            path[level_index - 1] = label
21            # Calculate the label's parent in the next higher level.
22            # Zigzag pattern means we have to invert the label within its level
23            label = ((1 << (level_index - 1)) + (1 << level_index) - 1 - label) >> 1
24            level_index -= 1
25
26        # Return the path that was constructed
27        return path
28
```

Java Solution

```
1 class Solution {
2     public List<Integer> pathInZigZagTree(int label) {
3         // Initialize the level to 1 and the start value of that level (x) to 1.
4         int level = 1;
5         int startOfLevel = 1;
6
7         // Determine the level of the tree where the label is located.
8         while ((startOfLevel * 2) <= label) {
9             startOfLevel *= 2;
10            ++level;
11        }
12
13        // Create a list to store the path from root to the label.
14        List<Integer> path = new ArrayList<>();
15
16        // Starting from the label's level, move up to the root.
17        for (int currentLevel = level; currentLevel > 0; --currentLevel) {
18            // Add the current label to the path.
19            path.add(label);
20
21            // Calculate the parent label in the previous level of a perfect binary tree,
22            // then adjust for the zigzag pattern.
23            int levelStart = (1 << (currentLevel - 1)); // Start of the current level
24            int levelEnd = (1 << currentLevel) - 1; // End of the current level
25            label = (levelStart + levelEnd - label) / 2;
26        }
27
28        // Since we've built the path from the bottom up, reverse it to get the correct order.
29        Collections.reverse(path);
30
31        return path;
32    }
33 }
34
```

C++ Solution

```
1 #include <vector>
2 #include <algorithm>
3
4 class Solution {
5 public:
6     // Method to find the path from the root to a given label in a zigzag labelled binary tree
7     vector<int> pathInZigZagTree(int label) {
8         // Initialize root level as 1, and depth as 1
9         int levelStartValue = 1, depth = 1;
10
11        // Calculate the depth of the given label. The depth increases while it is possible to go further down
12        while ((levelStartValue <= 1) <= label) {
13            levelStartValue <<= 1;
14            ++depth;
15        }
16
17        // Prepare a vector to store the path
18        vector<int> path;
19
20        // Loop from the level of the label to the root
21        for (; depth > 0; --depth) {
22            // Add current label to the path
23            path.push_back(label);
24            // Find the parent label. The operation calculates the opposite label in the same level and then finds the parent
25            label = ((1 << (depth - 1)) + (1 << depth) - 1 - label) >> 1;
26        }
27
28        // Reverse the path to start from the root
29        reverse(path.begin(), path.end());
30
31        // Return the path from the root to the label
32        return path;
33    }
34 };
35
```

Typescript Solution

```
1 // Import necessary TypeScript feature(s)
2 import { reverse } from 'lodash';
3
4 // Function to find the path from the root to a given label in a zigzag labeled binary tree
5 function pathInZigZagTree(label: number): number[] {
6     // Initialize root level value as 1, and depth as 1
7     let levelStartValue: number = 1;
8     let depth: number = 1;
9
10    // Calculate the depth of the given label
11    // The depth increases while it is possible to go further down
12    while ((levelStartValue <= 1) <= label) {
13        levelStartValue <<= 1;
14        depth++;
15    }
16
17    // Prepare an array to store the path
18    const path: number[] = [];
19
20    // Loop from the level of the label to the root
21    for (; depth > 0; depth--) {
22        // Add current label to the path
23        path.push(label);
24        // Find the parent label. This operation calculates the opposite label in the same level and then finds
25        // the parent in the previous level by performing integer division by 2
26        label = ((1 << (depth - 1)) + (1 << depth) - 1 - label) >> 1;
27    }
28
29    // Reverse the path so that it starts from the root
30    reverse(path);
31
32    // Return the path from the root to the label
33    return path;
34 }
35
36 // Example use of the function
37 const label: number = 14;
38 const pathFromRoot: number[] = pathInZigZagTree(label);
39 console.log(pathFromRoot); // Output the path to the console
40
```

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:

- The initial while loop runs as long as 2^i is less than or equal to `label`. Since the height of the tree increases logarithmically with respect to the `label`, this loop runs in $O(\log(\text{label}))$ time.
- 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 for a given `label` is $\log(\text{label})$, this loop will also execute $O(\log(\text{label}))$ times.
- Inside the second while loop, there are only constant time operations.

Thus, the overall time complexity of the function is $O(\log(\text{label})) + O(\log(\text{label})) = O(\log(\text{label}))$.

Space Complexity:

- 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 $O(\log(\text{label}))$, hence the space complexity for `ans` is $O(\log(\text{label}))$.
- A constant amount of auxiliary space is used for variables `x` and `i`.

Thus, the overall space complexity of the function is $O(\log(\text{label}))$.