Grokking the Coding Interview: Patterns for Coding Questions

56% completed

Search Course

Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)
- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)
- Level Order Successor (easy)
- Connect Level Order Siblings (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Depth First Search

- Introduction
- Binary Tree Path Sum (easy)
- All Paths for a Sum (medium)
- Sum of Path Numbers (medium)
- Path With Given Sequence (medium)
- Count Paths for a Sum (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Connect Level Order Siblings (medium)

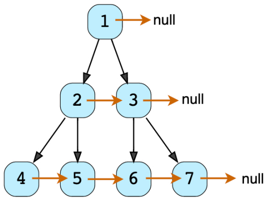
We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
  - Time complexity
  - Space complexity

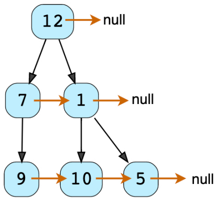
Problem Statement

Given a binary tree, connect each node with its level order successor. The last node of each level should point to a `null` node.

Example 1:



Example 2:



Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
class TreeNode {
  constructor(val) {
    this.val = val;
    this.left = null;
    this.right = null;
    this.next = null;
  }

  // level order traversal using 'next' pointer
  print_level_order() {
    console.log("Level order traversal using 'next' pointer: ");
    let nextLevelRoot = this;
    while (nextLevelRoot !== null) {
      let current = nextLevelRoot;
      nextLevelRoot = null;
      while (current != null) {
        process.stdout.write(`${current.val} `);
        if (nextLevelRoot === null) {
          if (current.left !== null) {
            nextLevelRoot = current.left;
          } else if (current.right !== null) {
            nextLevelRoot = current.right;
          }
        }
        current = current.next;
      }
      console.log();
    }
```

RUN | SAVE | RESET

Solution

Problem Challenge 2

Solution Review: Problem Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

Pattern: Modified Binary Search

● Introduction

● Order-agnostic Binary Search (easy)

● Ceiling of a Number (medium)

Next Letter (medium)

Number Range (medium)

Search in a Sorted Infinite Array (medium)

Minimum Difference Element (medium)

Bitonic Array Maximum (easy)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

Pattern: Bitwise XOR

Introduction

Single Number (easy)

Two Single Numbers (medium)

Complement of Base 10 Number (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Pattern: Top 'K' Elements

● Introduction

● Top 'K' Numbers (easy)

Kth Smallest Number (easy)

'K' Closest Points to the Origin (easy)

Connect Ropes (easy)

Top 'K' Frequent Numbers (medium)

Frequency Sort (medium)

Kth Largest Number in a Stream (medium)

'K' Closest Numbers (medium)

Maximum Distinct Elements (medium)

Sum of Elements (medium)

Rearrange String (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Problem Challenge 3

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

This problem follows the Binary Tree Level Order Traversal pattern. We can follow the same **BFS** approach. The only difference is that while traversing a level we will remember the previous node to connect it with the current node.

Code

Here is what our algorithm will look like; only the highlighted lines have changed:

Java | Python3 | C++ | JS

```
const Deque = require('./collections/deque'); //http://www.collectionsjs.com

class TreeNode {
  constructor(val) {
    this.val = val;
    this.left = null;
    this.right = null;
    this.next = null;
  }

  // level order traversal using 'next' pointer
  print_level_order() {
    console.log("Level order traversal using 'next' pointer: ");
    let nextLevelRoot = this;
    while (nextLevelRoot !== null) {
      let current = nextLevelRoot;
      nextLevelRoot = null;
      while (current != null) {
        process.stdout.write(`${current.val} `);
        if (nextLevelRoot === null) {
          if (current.left !== null) {
            nextLevelRoot = current.left;
          } else if (current.right !== null) {
            nextLevelRoot = current.right;
          }
        }
        current = current.next;
      }
```

RUN SAVE RESET

Close

Output 3.675s

```
Level order traversal using 'next' pointer:
12
7 1
9 10 5
```

Time complexity

The time complexity of the above algorithm is $O(N)$, where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once.

Space complexity

The space complexity of the above algorithm will be $O(N)$, which is required for the queue. Since we can have a maximum of $N/2$ nodes at any level (this could happen only at the lowest level), therefore we will need $O(N)$ space to store them in the queue.

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. See how ⓘ

← Back

Level Order Successor (easy)

Next →

Problem Challenge 1

☑ Mark as Completed

Report an Issue Ask a Question