Solution: Populating Next Right Pointers in Each Node

Let's solve the Populating Next Right Pointers in the Each Node problem using the Tree Breadth-first Search pattern.

We'll cover the following

Statement

Given a binary tree, connect all nodes of the same hierarchical level. We need to connect them from left to right, so that the next pointer of each node points to the node on its immediate right. The next pointer of the right-most node at each level will be NULL.

For this problem, each node in the binary tree has one additional pointer (the `next` pointer) along with the `left` and `right` pointers.

Constraints:

- The number of nodes in the tree is in the range $[0, 2^{12} - 1]$.
- $-1000 \leq$ `Node.data` ≤ 1000

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

The naive solution would be to use a queue for the level order traversal of our binary tree. Since we also need to keep track of the node's level, we can push a (node, level) pair to the queue. However, this approach isn't efficient since we only need the nodes at a certain level. Since we keep popping the old ones to push their children to the queue, we'll have to use a separate data structure to keep track of the same level nodes.

The overall time complexity of the algorithm becomes $O(n)$, where n is the number of nodes. The space complexity is also $O(n)$.

Optimized approach using tree breadth-first search

Since we need to connect all nodes of the same hierarchical level, this problem falls naturally under the tree breadth-first search pattern. We'll perform a level order traversal and connect the nodes to their siblings.

We need to connect all nodes at each hierarchical level with the `next` pointer.

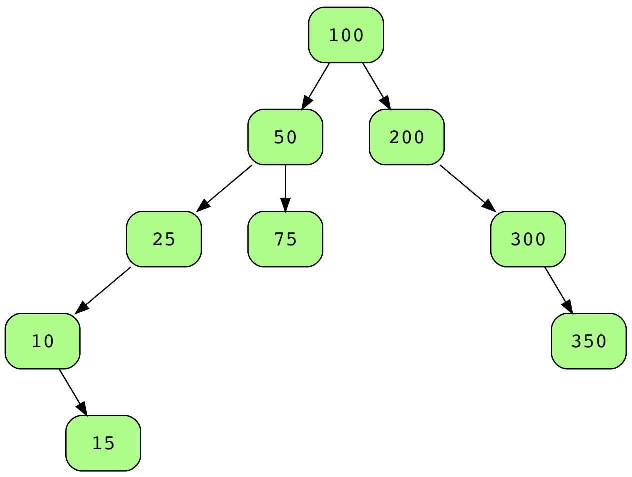
Here's how the algorithm works:

1. Start at level 0, where the root of the binary tree is present.
2. Now perform the following steps at the current level x:
   - We begin at the given starting node at level x, which already has all nodes connected.
   - Then traverse the nodes at level x from left to right while connecting their children at the next level $(x + 1)$ with the `next` pointers.
   - If the current node has both the left and right child, connect them with each other and the previously connected nodes.
   - Otherwise, if the current node only has a left or right child, connect it with the previously connected nodes.
3. Now that the nodes at level $x + 1$ are connected, we use the head node (the left-most node of $x + 1$ level) and perform the same process as step 2 on the next level.
4. Repeat steps 2 and 3 until we exhaust all levels of the binary tree.

To connect the nodes at a level given the head of that level, we use a `prev` pointer along with `current` to keep track of the previously connected nodes.

Let's understand this algorithm by connecting nodes at each level for the example given above:

Initial state of the tree.



1 of 22

Let's look at the code for this solution below:

Java

main.java

EduBinaryTree.java

EduTreeNode.java

```
class ConnectSameLevelNodes {
    // Helper function to connect all children nodes at the next level
    private static EduTreeNode<Integer> connectNextLevel(EduTreeNode<Integer>
        // Declaring the necessary pointers
        EduTreeNode<Integer> current = head;
        EduTreeNode<Integer> nextLevelHead = null;
        EduTreeNode<Integer> prev = null;

        while (current != null) {
            if (current.left != null && current.right != null) {
                // If both current node children are not null
                // then connect them with each other and previous
                // nodes in the same level.
                if (nextLevelHead == null) {
                    nextLevelHead = current.left;
                }
                current.left.next = current.right;

                if (prev != null) {
                    prev.next = current.left;
                }
                prev = current.right;
            } else if (current.left != null) {
                // If only the left child node is not null
                // then only connect it with previous same level nodes
```

Populating Next Right Pointers in Each Node

Solution summary

To recap, the solution to this problem can be divided into the following parts:

1. Perform a breadth-first search on the tree starting from the root node.
2. If a node has both left and right children, first connect them with each other and then with the previous nodes at the same level.
3. If a node has only one child, connect it with the previous nodes at the same level.
4. Set the next pointer of the right-most node to NULL and move to the next level.
5. Repeat the steps above until all nodes have been visited.

Time complexity

The time complexity of this solution is linear, $O(n)$, where n is the number of nodes in the binary tree.

Space complexity

The space complexity of this solution is constant, $O(1)$.

← Back
Populating Next Right...

Next →
Vertical Order Travers...

✓ Mark as Completed