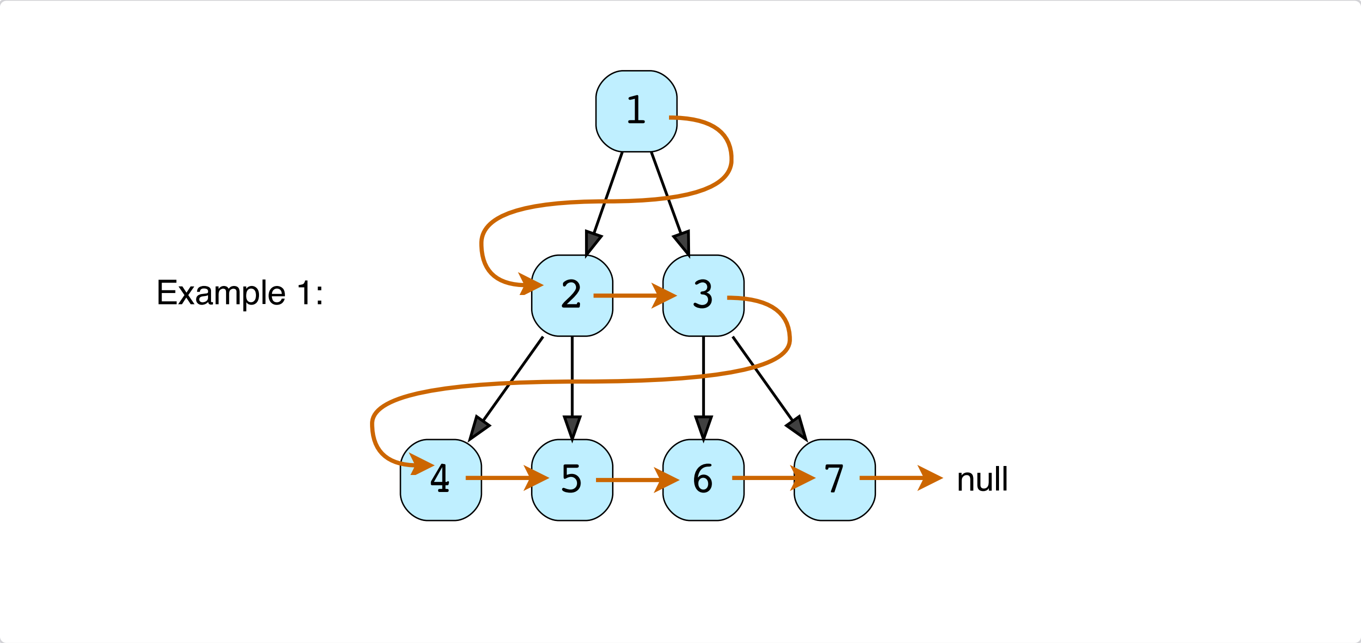
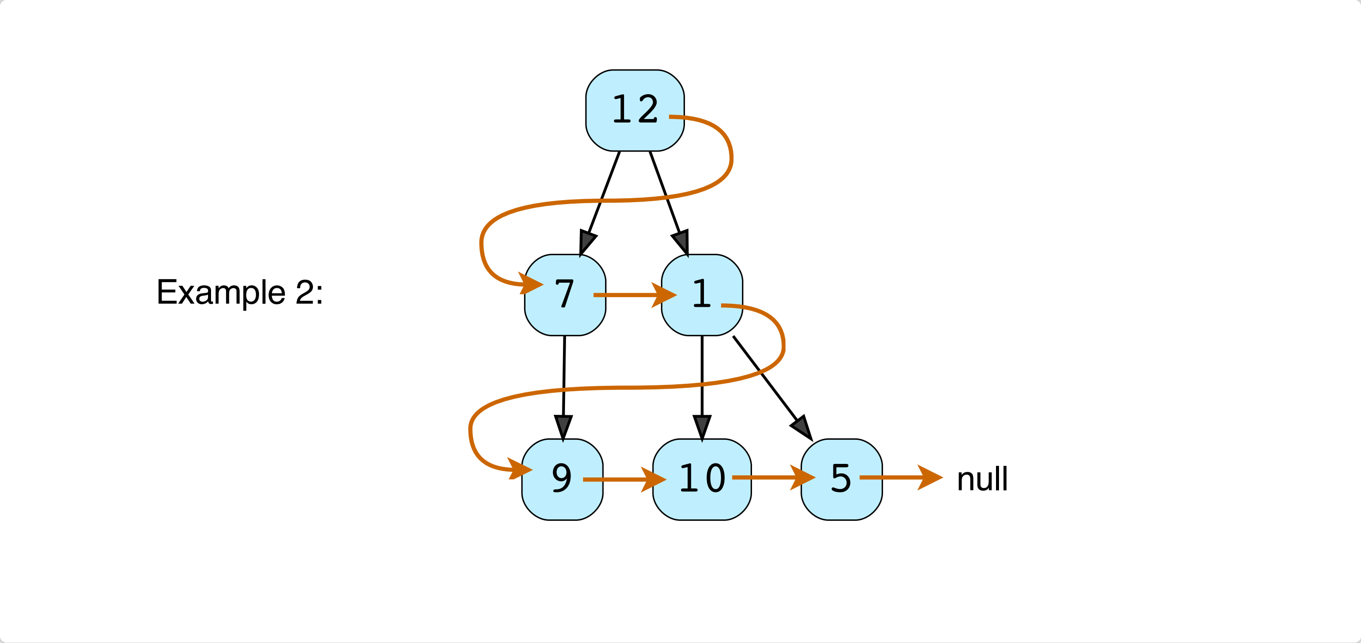
Solution Review: Problem Challenge 1

We'll cover the following

Connect All Level Order Siblings (medium)#

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





Solution#

This problem follows the Binary Tree Level Order Traversal (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5726607939469312/) pattern. We can follow the same **BFS** approach.

The only difference will be that while traversing we will remember (irrespective of the level) 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

```
import java.util.*;

class TreeNode {
  int val;
  TreeNode left;
  TreeNode right;
  TreeNode next;

  TreeNode(int x) {
    val = x;
    left = right = next = null;
  }

  // tree traversal using 'next' pointer
  public void printTree() {
    TreeNode current = this;
    System.out.print("Traversal using 'next' pointer: ");
    while (current != null) {
      System.out.print(current.val + " ");
      current = current.next;
    }
  }
};

class ConnectAllSiblings {
  public static void connect(TreeNode root) {
    if (root == null)
      return;
    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    TreeNode currentNode = null, previousNode = null;
    while (!queue.isEmpty()) {
      currentNode = queue.poll();
      if (previousNode != null)
        previousNode.next = currentNode;
      previousNode = currentNode;

      // insert the children of current node in the queue
      if (currentNode.left != null)
        queue.offer(currentNode.left);
      if (currentNode.right != null)
        queue.offer(currentNode.right);
    }
  }

  public static void main(String[] args) {
    TreeNode root = new TreeNode(12);
    root.left = new TreeNode(7);
    root.right = new TreeNode(1);
    root.left.left = new TreeNode(9);
    root.right.left = new TreeNode(10);
    root.right.right = new TreeNode(5);
    ConnectAllSiblings.connect(root);
    root.printTree();
  }
}
```

Run Save Reset

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.

← Back

Problem Challenge 1

Next →

Problem Challenge 2

☑ Mark as Completed

Report an Issue