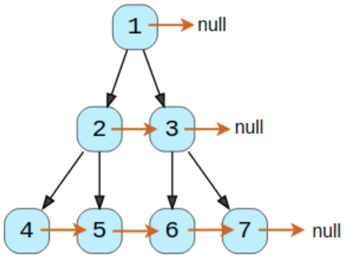
Connect Level Order Siblings (medium)

We'll cover the following

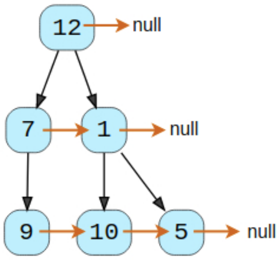
Problem Statement

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:

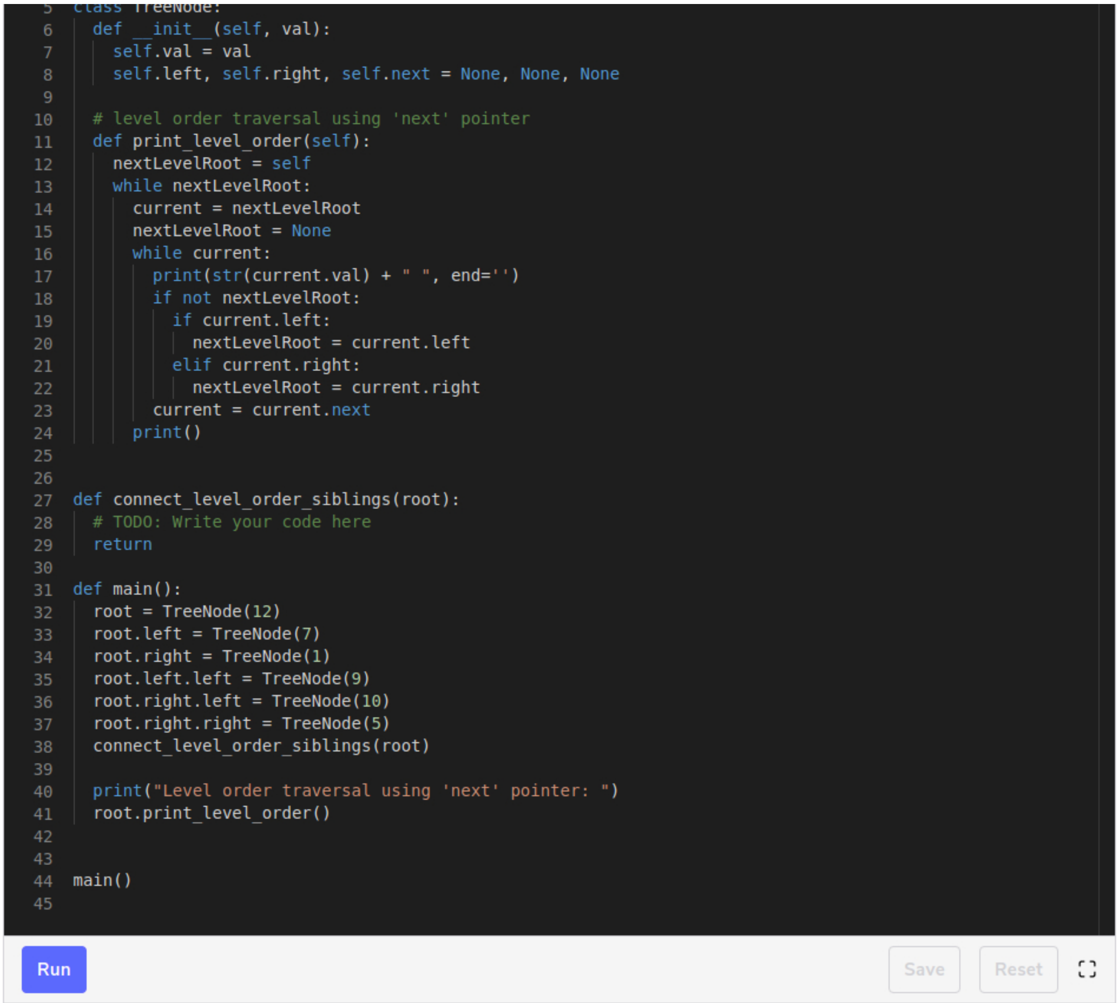


Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
from __future__ import print_function
from collections import deque


class TreeNode
```



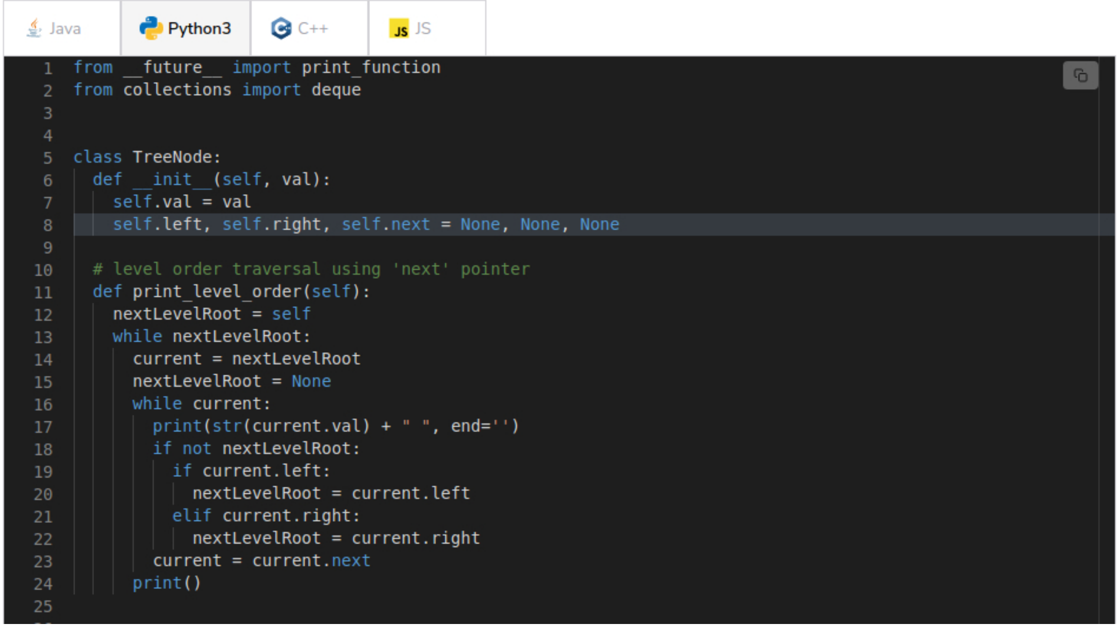
```python
class TreeNode:
  def __init__(self, val):
    self.val = val
    self.left, self.right, self.next = None, None, None

  # level order traversal using 'next' pointer
  def print_level_order(self):
    nextLevelRoot = self
    while nextLevelRoot:
      current = nextLevelRoot
      nextLevelRoot = None
      while current:
        print(str(current.val) + " ", end='')
        if not nextLevelRoot:
          if current.left:
            nextLevelRoot = current.left
          elif current.right:
            nextLevelRoot = current.right
        current = current.next
      print()


def connect_level_order_siblings(root):
  # TODO: Write your code here
  return

def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
  root.right = TreeNode(1)
  root.left.left = TreeNode(9)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  connect_level_order_siblings(root)

  print("Level order traversal using 'next' pointer: ")
  root.print_level_order()


main()
```

Solution

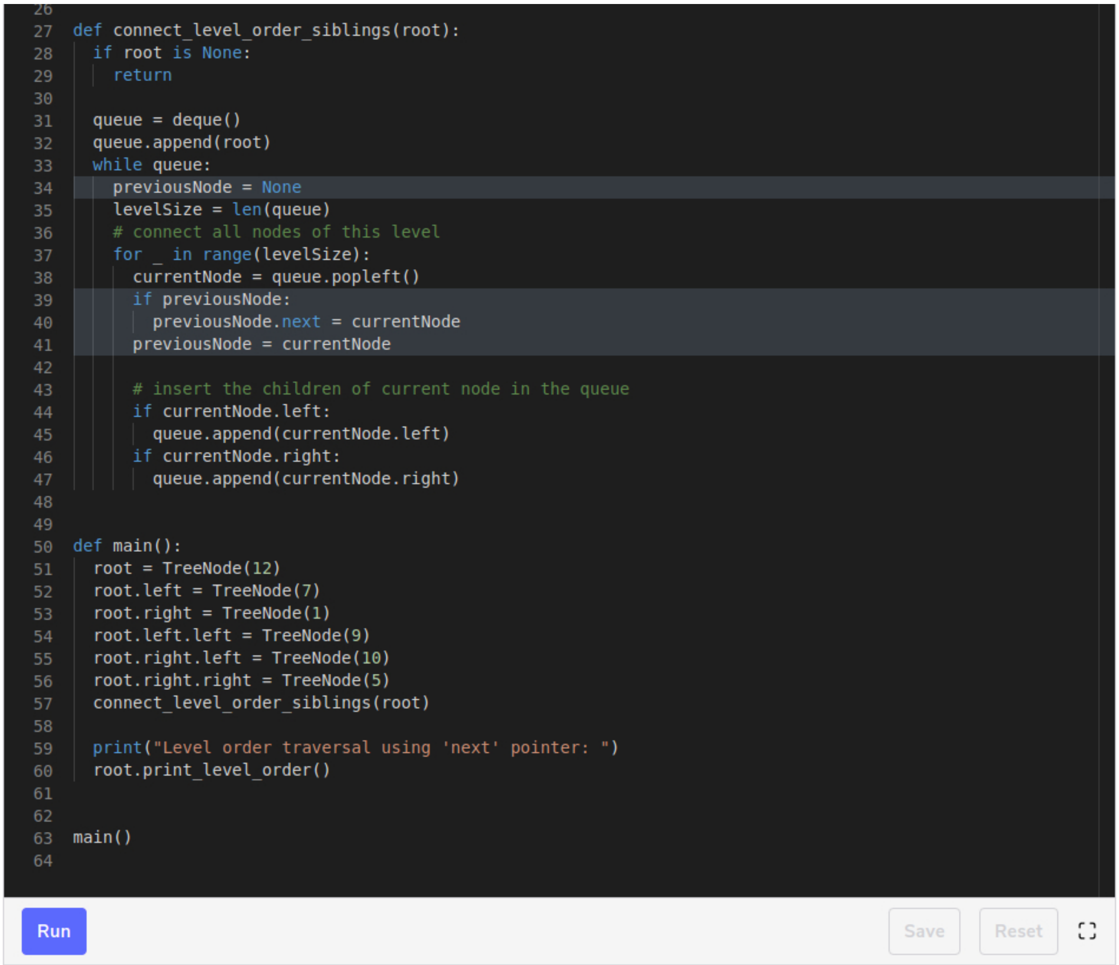
This problem follows the Binary Tree Level Order Traversal pattern. We can follow the same **BFS** approach. The only difference is that while traversing a level we will remember the previous node to connect it with the current node.

Code

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



```python
from __future__ import print_function
from collections import deque


class TreeNode:
  def __init__(self, val):
    self.val = val
    self.left, self.right, self.next = None, None, None

  # level order traversal using 'next' pointer
  def print_level_order(self):
    nextLevelRoot = self
    while nextLevelRoot:
      current = nextLevelRoot
      nextLevelRoot = None
      while current:
        print(str(current.val) + " ", end='')
        if not nextLevelRoot:
          if current.left:
            nextLevelRoot = current.left
          elif current.right:
            nextLevelRoot = current.right
        current = current.next
      print()
```



```
def connect_level_order_siblings(root):
  if root is None:
    return

  queue = deque()
  queue.append(root)
  while queue:
    previousNode = None
    levelSize = len(queue)
    # connect all nodes of this level
    for _ in range(levelSize):
      currentNode = queue.popleft()
      if previousNode:
        previousNode.next = currentNode
      previousNode = currentNode

      # insert the children of current node in the queue
      if currentNode.left:
        queue.append(currentNode.left)
      if currentNode.right:
        queue.append(currentNode.right)


def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
  root.right = TreeNode(1)
  root.left.left = TreeNode(9)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  connect_level_order_siblings(root)

  print("Level order traversal using 'next' pointer: ")
  root.print_level_order()


main()
```

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
Level Order Successor (easy)

Next →
Problem Challenge 1

✓ Completed

Report an Issue Ask a Question