educative

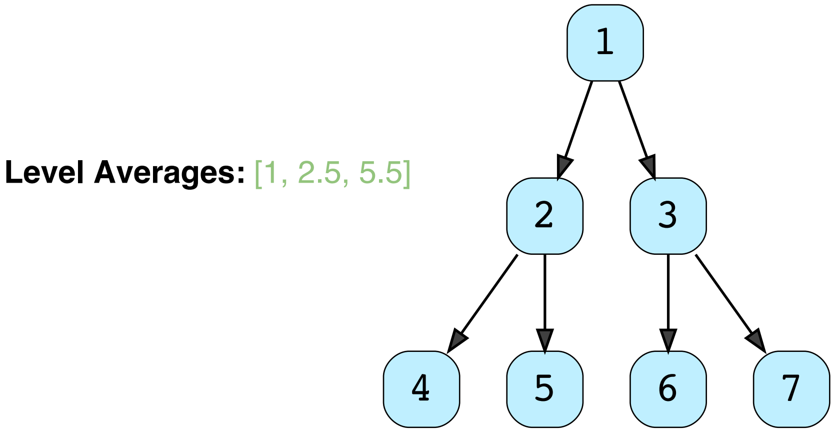
Level Averages in a Binary Tree (easy)

We'll cover the following

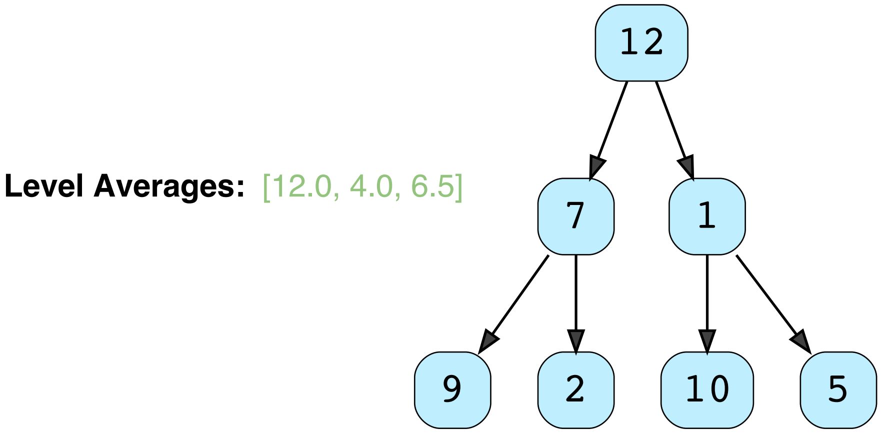
Problem Statement

Given a binary tree, populate an array to represent the **averages of all of its levels**.

Example 1:



Example 2:

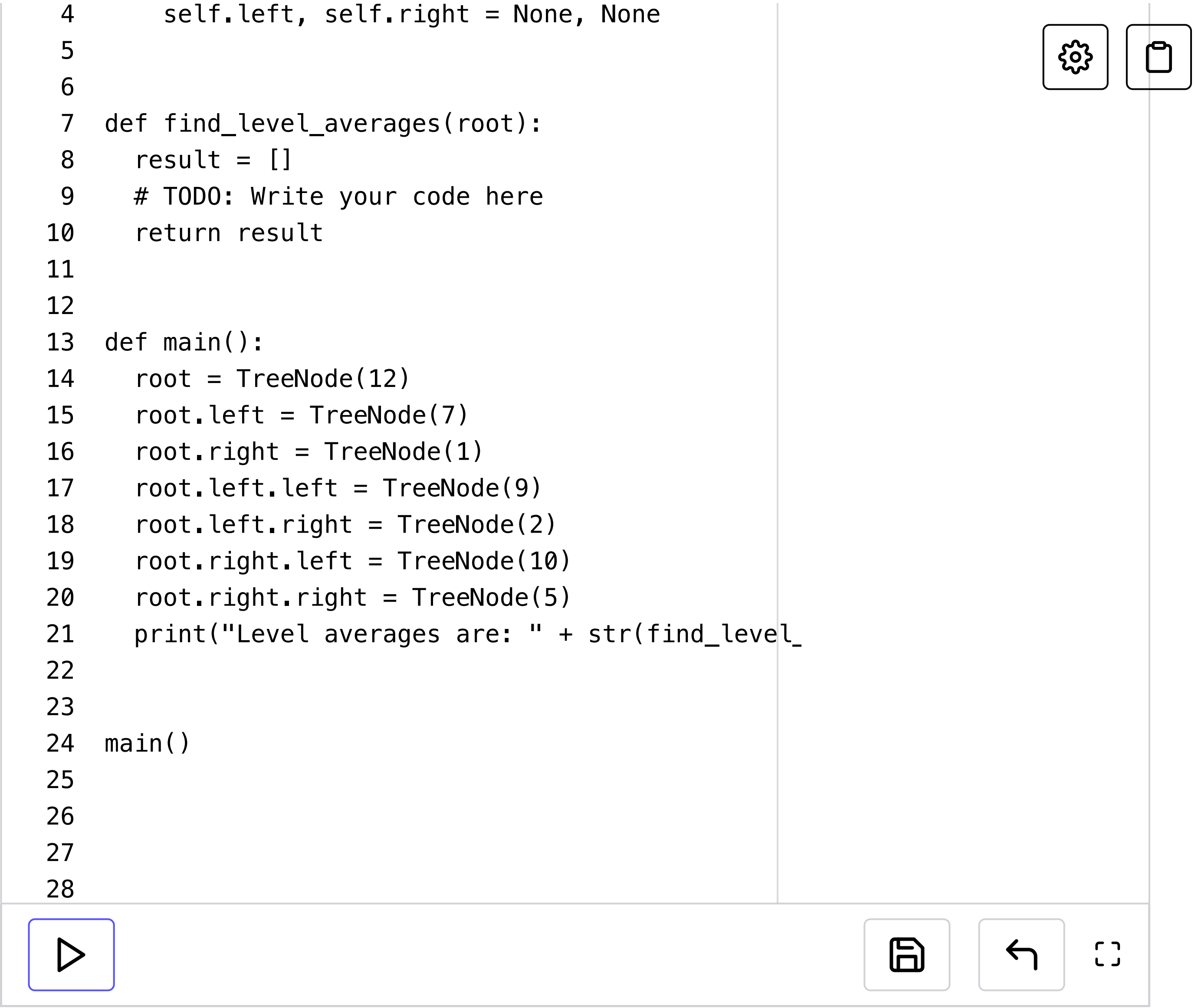


Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
class TreeNode:
  def __init__(self, val):
    self.val = val
```



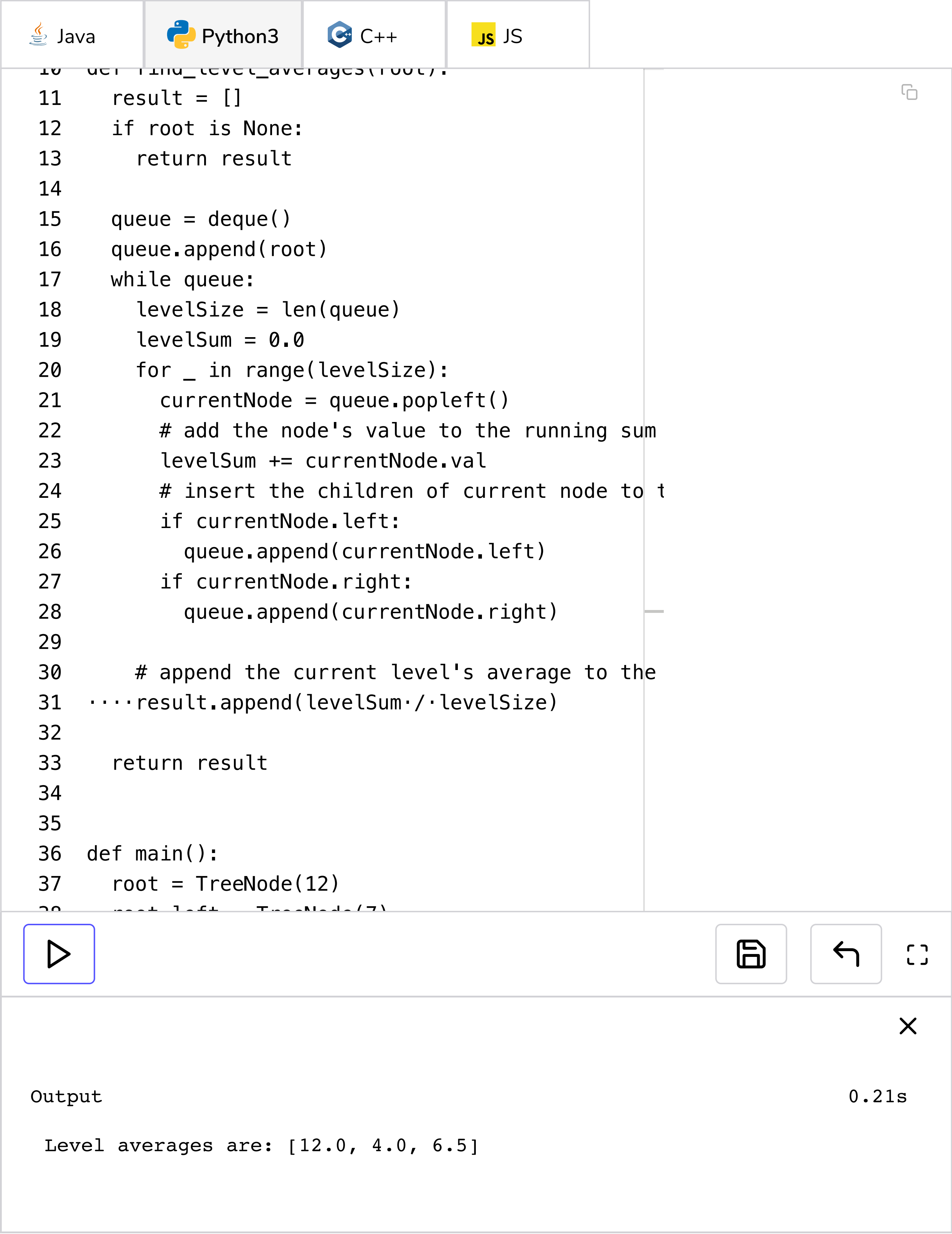
```
    self.left, self.right = None, None


def find_level_averages(root):
  result = []
  # TODO: Write your code here
  return result


def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
  root.right = TreeNode(1)
  root.left.left = TreeNode(9)
  root.left.right = TreeNode(2)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  print("Level averages are: " + str(find_level_


main()
```

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 instead of keeping track of all nodes of a level, we will only track the running sum of the values of all nodes in each level. In the end, we will append the average of the current level to the result array.

Code

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



```python
def find_level_averages(root):
  result = []
  if root is None:
    return result

  queue = deque()
  queue.append(root)
  while queue:
    levelSize = len(queue)
    levelSum = 0.0
    for _ in range(levelSize):
      currentNode = queue.popleft()
      # add the node's value to the running sum
      levelSum += currentNode.val
      # insert the children of current node to t
      if currentNode.left:
        queue.append(currentNode.left)
      if currentNode.right:
        queue.append(currentNode.right)

    # append the current level's average to the
    result.append(levelSum / levelSize)

  return result


def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
```

```
Output                                                    0.21s

 Level averages are: [12.0, 4.0, 6.5]
```

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.

Similar Problems

Problem 1: Find the largest value on each level of a binary tree.

Solution: We will follow a similar approach, but instead of having a running sum we will track the maximum value of each level.

```
maxValue = max(maxValue, currentNode.val)
```

← Back

Zigzag Traversal (medium)

Next →

Minimum Depth of a Binary Tree (easy)