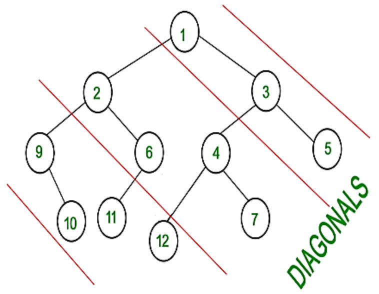
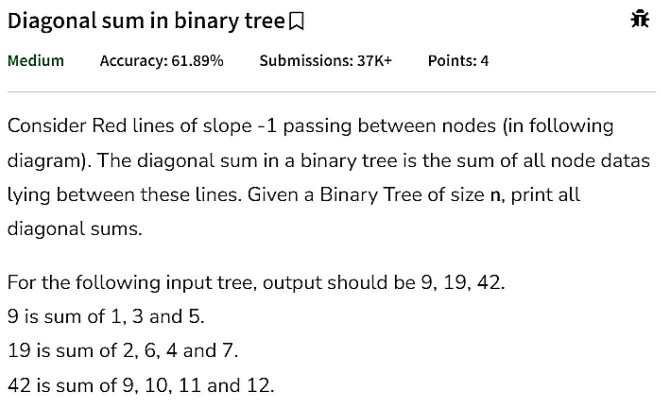
PROBLEM



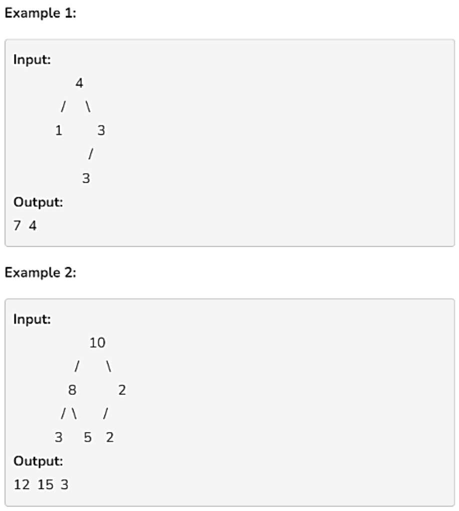


Diagonal sum in binary tree

Medium Accuracy: 61.89% Submissions: 37K+ Points: 4

Consider Red lines of slope -1 passing between nodes (in following diagram). The diagonal sum in a binary tree is the sum of all node datas lying between these lines. Given a Binary Tree of size **n**, print all diagonal sums.

For the following input tree, output should be 9, 19, 42.
9 is sum of 1, 3 and 5.
19 is sum of 2, 6, 4 and 7.
42 is sum of 9, 10, 11 and 12.

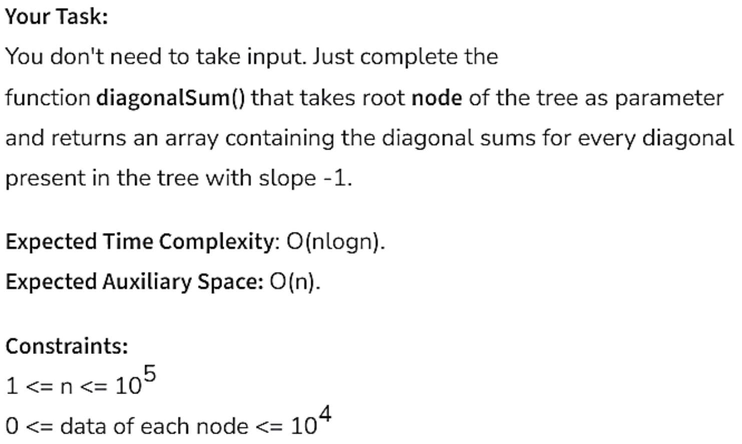


Example 1:

```
Input:
         4
       /   \
      1     3
           /
          3
Output:
7 4
```

Example 2:

```
Input:
           10
         /    \
        8      2
       / \    /
      3   5  2
Output:
12 15 3
```



Your Task:
You don't need to take input. Just complete the function **diagonalSum()** that takes root **node** of the tree as parameter and returns an array containing the diagonal sums for every diagonal present in the tree with slope -1.

Expected Time Complexity: O(nlogn).
Expected Auxiliary Space: O(n).

Constraints:
1 <= n <= 10^5
0 <= data of each node <= 10^4

CODE

```
#User function Template for python3

'''
# Node Class:
class Node:
    def __init__(self,val):
        self.data = val
        self.left = None
        self.right = None
'''
```

```
class Solution:
    #Complete the function below
    def diagonalSum(self, root):
        min_level, max_level = 0, 0
        level_sums = {}  # Dictionary to store sums at each level
        queue = deque()
        queue.append([root, 0])  # Adding root node with level 0 to the queue

        while queue:
            node, level = queue.popleft()
            level_sums[level] = level_sums.get(level, 0) + node.data
            min_level, max_level = min(min_level, level), max(max_level, level)
            if node.left: queue.append([node.left, level - 1])  # Adding left child with
level decreased by 1
            if node.right: queue.append([node.right, level])  # Adding right child with
same level

        # Returning diagonal sums from max level to min level
        return [level_sums[i] for i in range(max_level, min_level-1,-1)]
```

EXPLANATION

Firstly, we have a class named **Solution** which contains a method **diagonalSum** that takes a **root** node of a binary tree as its input.

```
class Solution:
    # Complete the function below
    def diagonalSum(self, root):
```

Within this method, the code initializes **min_level** and **max_level** variables to keep track of the minimum and maximum levels encountered during the traversal of the binary tree.

```
        min_level, max_level = 0, 0
```

level_sums is a dictionary used to store the sums at each level of the binary tree.

```
        level_sums = {}  # Dictionary to store sums at each level
```

We are using a **queue** data structure to perform a level-order traversal of the binary tree. We start with the root node and its level (which is initially 0), and then we append it to the queue.

```
        queue = deque()
        queue.append([root, 0])  # Adding root node with level 0 to the queue
```

Now, we iterate through the elements in the queue until it's empty.

while queue:

Within the loop, we dequeue a node along with its level from the queue.

```
node, level = queue.popleft()
```

Then, we update the sum at the current level in the **level_sums** dictionary.

```
level_sums[level] = level_sums.get(level, 0) + node.data
```

We update **min_level** and **max_level** to keep track of the minimum and maximum levels encountered during the traversal.

```
min_level, max_level = min(min_level, level), max(max_level, level)
```

Next, we enqueue the left child (if it exists) with a level decreased by 1 and the right child (if it exists) with the same level.

```
if node.left: queue.append([node.left, level - 1])  # Adding left child with level decreased by 1
if node.right: queue.append([node.right, level])  # Adding right child with same level
```

Once the traversal is complete, we construct and return the diagonal sums from the maximum level encountered to the minimum level encountered.

```
# Returning diagonal sums from max level to min level
return [level_sums[i] for i in range(max_level, min_level-1,-1)]
```

Consider the following binary tree:

```
    1
   / \
  2   3
 / \   \
4   5   6
```

Here, 1 is the root node, 2 and 3 are its children, 2 has children 4 and 5, and 3 has a child 6.

Let's represent this tree in code:

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

```
# Constructing the binary tree
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.right = TreeNode(6)
```

Now, let's create an instance of the Solution class and call the diagonalSum function with the root of our tree:

```
solution = Solution()
result = solution.diagonalSum(root)
print(result)
```

This should output the diagonal sums of the tree. Let's analyze how the function works:

We start at the root node 1 with level 0.
We enqueue [1, 0] into the queue.
We dequeue [1, 0], update the sum at level 0 to 1, and enqueue [2, -1] (left child) and [3, 0] (right child).
We dequeue [2, -1], update the sum at level -1 to 2, and enqueue [4, -2] (left child) and [5, -1] (right child).
We dequeue [3, 0], update the sum at level 0 to 3, and enqueue [6, 0] (right child).
We dequeue [4, -2], update the sum at level -2 to 4.
We dequeue [5, -1], update the sum at level -1 to 5.
We dequeue [6, 0], update the sum at level 0 to 6.
The traversal is complete.
The diagonal sums are calculated from the bottom-left to the top-right diagonal.
So, the output would be [4, 7, 6], representing the sums at levels -2, -1, and 0 respectively.