CS566Lab7-2024

October 23, 2024

0.1 Lab 7. Introduction to algorithms

This is seventh Lab for CS 566. This problem was given in lecture.

0.2 Task 1. Solve the problem "Level Order Traversal" from https://leetcode.com/problems/binary-tree-level-order-traversal/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[1]:
```python
from collections import deque
from typing import Optional, List
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []

        res = []
        q = deque([root])

        while q:
            level = []
            for i in range(len(q)):
                node = q.popleft()
                level.append(node.val)
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)
            res.append(level)
```

```
        return res
```

0.2.1 Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

[2]:
```
#test_case_1
root = TreeNode(3, TreeNode(9), TreeNode(20, TreeNode(15), TreeNode(7)))
expected = [[3],[9,20],[15,7]]
actual = Solution().levelOrder(root)
assert actual==expected, "Mistake in test case 1"
print("OK")
```

```
OK
```

0.2.2 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n)

Time Analysis: O(n)

0.3 Task 2. Solve the problem "Search in BST" from https://leetcode.com/problems/search-in-a-binary-search-tree/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[3]:
```
from typing import Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def searchBST(self, root: Optional[TreeNode], val: int) ->␣
 ↪Optional[TreeNode]:
        if not root:
            return root

        if val>root.val:
            return self.searchBST(root.right,val)
        elif val<root.val:
            return self.searchBST(root.left,val)
        else:
            return root
```

[4]:
```
#test_case_1
root = root = TreeNode(5, TreeNode(1), TreeNode(4, TreeNode(3), TreeNode(6)))
expected = 6
actual = Solution().searchBST(root, 6)
assert actual.val==expected, "Mistake in test case 1"

#test_case_2
root =  TreeNode(5, TreeNode(1), TreeNode(4, TreeNode(3),␣
 ↪TreeNode(6)))#[5,1,4,null,null,3,6]
expected = None
actual = Solution().searchBST(root, 7)
assert actual==expected, "Mistake in test case 2"

print('OK')
```

```
OK
```

0.3.1 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(h) - where h is height of tree

Time Analysis: O(h) - where h is height of tree

0.4 Task 3. Solve the problem "Validate Binary Search Tree" from https://leetcode.com/problems/validate-binary-search-tree/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[5]:
```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def isValidBST(self, root: Optional[TreeNode], no_greater_than = ␣
 ↪float('inf'), no_less_than = float('-inf')) -> bool:
        if root is None:
            return True

        if root.val>=no_greater_than or root.val<=no_less_than:
            return False

        min_val = min(no_greater_than, root.val)
```

```
        max_val = max(no_less_than, root.val)

        return self.isValidBST(root.left,min_val, no_less_than) and self.
 ↪isValidBST(root.right,no_greater_than, max_val)
```

[6]:
```
#test_case_1
root = TreeNode(2, TreeNode(1), TreeNode(3))
expected = True
actual = Solution().isValidBST(root)
assert actual==expected, "Mistake in test case 1"


root =  TreeNode(5, TreeNode(1), TreeNode(4, TreeNode(3),␣
 ↪TreeNode(6)))#[5,1,4,null,null,3,6]
expected = False
actual = Solution().isValidBST(root)
assert actual==expected, "Mistake in test case 2"

print('OK')
```

```
OK
```

0.4.1 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n)

Time Analysis: O(n)