100

ryan

Other

PDF generated at: 6 Aug 2024 16:41:12 UTC

View this report on HackerRank

Score

100% • 80 / 80

scored in CodePath TIP101: Unit 8 Assessment, Version A - Summer 2024 in 28 min 27 sec on 6 Aug 2024 09:10:41 PDT

Candidate Information

Email	concepting@protonmail.com
Test	CodePath TIP101: Unit 8 Assessment, Version A - Summer 2024
Candidate Packet	View
Taken on	6 Aug 2024 09:10:41 PDT
Time taken	28 min 27 sec/ 90 min
Work Experience	< 1 years
Invited by	CodePath

Suspicious Activity detected

Code similarity

Code similarity
1 question

Skill Distribution

There is no associated skills data that can be shown for this assessment

Tags Distribution

There is no associated tags data that can be shown for this assessment

Questions

Status	No.	Question	Time Taken	Skill	Score
✓	1	Tree Structure Multiple Choice	2 min	-	5/5
✓	2	Fix the Binary Tree Search Function Multiple Choice	7 min 41 sec	-	5/5

✓	3	Mystery Output Multiple Choice	1 min 49 sec	-	5/5
✓	4	Time Complexity Multiple Choice	59 sec	-	5/5
✓	5	Insert Node into a Binary Search Tree Coding	2 min 5 sec	-	20/20
✓	6	Sum the Nodes Coding	3 min 5 sec	-	20/20
✓	7	Remove Node from a Binary Search Tree Coding	10 min 41 sec	-	20/20

1. Tree Structure

✓ Correct

Multiple Choice

Question description

Given the following code, which of the following best represents the values of the tree with root `root`.

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

a = Node('a')
x = Node('x')
y = Node('y')
e = Node('e')
m = Node('m')
```

```
p = Node('p')

a.left = x
a.right = y

x.left = e
x.right = m

y.right = p

root = a
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ( ) <pre> <code class="language-python"> root / \ / \ x y / \ \ e m p</code></pre>
 <!-- notionvc: 335ce73b-fcbe-45c0-9041-d7f7f5efe34b --><!-- notionvc: 43cffeaa-2d37-4866-90ce-a0371fad32e6 -->
- (✓) **<pre> <code class="language-python"> a / \ / \ x y / \ \ e m p</code></pre>
 <!-- notionvc: 328084e6-4df2-4ae9-834b-5e008b6dce37 --> **
- ( ) <pre> <code class="language-python"> a / \ / \ x y / \ \ m e p</code></pre>
 <!-- notionvc: a3c29946-ba02-4e14-9e8b-f5aa1d00f0eb --><!-- notionvc: ad2c3bdc-8b34-4881-996a-e480c9a25a6a --><!-- notionvc: 169fa24e-3d44-491e-a51c-1156fbd4cbc7 --><!-- notionvc: 10453584-991a-4f8b-a411-722a914c99cb -->
- ( ) <pre> <code class="language-python"> a / \ / \ x y / \ / e m p </code></pre>
 <!-- notionvc: 9befdc5c-dc6d-4f6d-9268-e53ece276445 --><!-- notionvc: 04f9b35e-26c4-40e8-95e0-4b1c535fd683 --><!-- notionvc: 28a7e321-eafd-4ce4-9c2e-1d3e932376dc --><!--

notionvc: 0832d073-fd74-457c-90db-735f13c730c7 --><!-- notionvc: a71bb310-8779-49dc-9de1-6dbaafe45317 --></p>

No comments.

2. Fix the Binary Tree Search Function

Correct

Multiple Choice

Question description

The following function `find()` should take in the `root` of a binary tree and a value `target` and return `True` if there is a node with value `target` in the tree and `False` otherwise.
However, it has a bug!

Choose the option that fixes the bug and correctly implements the `find()` function.

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

def find(root, target):
    if not root:
        return False

    if root.val == target:
        return True

    return find(root.left, target) and find(root.right, target)

# Example Usage
# Define the tree
#    1
#   / \
#  2   3
```

```
# / \
#4   5
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)

output = find(root, 3)  # Expected Output: True, Actual Output: False
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [x] **<pre> <code class="language-python">class TreeNode: def __init__(self, val=0, left=None, right=None): self.val = val self.left = left self.right = right def find(root, target): if not root: return False if root.val == target: return True # replace 'and' keyword with 'or' return find(root.left, target) or find(root.right, target)</code></pre>   **

- [ ] <pre> <code class="language-python">class TreeNode: def __init__(self, val=0, left=None, right=None): self.val = val self.left = left self.right = right def find(root, target): if not root: return False # Call helpers before checking if current node value matches target find(root.left, target) find(root.right, target) if root.val == target: return True </code></pre>

- [ ] <pre> <code class="language-python">class TreeNode: def __init__(self, val=0, left=None, right=None): self.val = val self.left = left self.right = right def find(root, target): if not root: return False # Call helpers inside if statement if root.val == target: return find(root.left, target) or find(root.right, target)</code></pre>

<pre> <code class="language-python">class TreeNode: def __init__(self, val=0, left=None, right=None): self.val = val self.left = left self.right = right # Move recursive logic to a helper function def find_helper(root, target): if root.val == target: return True return find(root.left, target) and find(root.right, target) def find(root, target): if not root: return False return find_helper(root, target)</code></pre>

No comments.

3. Mystery Output

Correct

Multiple Choice

Question description

Given the following code, what is the value of output?

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

def mystery_function(root):
    if not root:
        return 0

    return 1 + mystery_function(root.left) + mystery_function(root.right)


# Define the tree
#     1
#    / \
#   2   3
#  /
# 4
```

```
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)

output = mystery_function(root)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ 1
- ○ 2
- ○ 3
- ● 4 ✓

No comments.

4. Time Complexity

Correct

Multiple Choice

Question description

What is the time complexity of `mystery_function()`?

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

def mystery_function(root):
    if not root:
        return 0

    return 1 + mystery_function(root.left) + mystery_function(root.right)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ O(1)
- ○ O(log n)
- ● **O(n)** ✓
- ○ $O(n^2)$

No comments.

5. Insert Node into a Binary Search Tree

Correct

Coding

Question description

Given the `root` of a binary search tree, insert a node with the value `val` into the tree. Return the `root`.

Example Input Tree:

```
  3
 / \
2  4
```

Input: root = 3, val = 5
Expected Output: root = 3

```
  3
 / \
2  4
   \
    5
```

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast


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

```
        self.right = right

#
# Complete the 'insert_node' function below.
#
# The function is expected to return a TreeNode.
# The function accepts following parameters:
#  1. TreeNode root
#  2. INTEGER val
#

def insert_node(root, val):
    # Write your code here
    if not root:
        return TreeNode(val)
    if val < root.val:
        root.left = insert_node(root.left, val)
    else:
        root.right = insert_node(root.right, val)
    return root


if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    def make_tree(tree_tup):
        if tree_tup is None:
            return None # Base case
        elif len(tree_tup) != 3:
            print("Invalid input: ", tree_tup)
            return None # Invalid case
        # Happy case
        return TreeNode(tree_tup[0], make_tree(tree_tup[1]),
make_tree(tree_tup[2]))

    def unmake_tree(root):
        if root is None:
            return None
        return (root.val, unmake_tree(root.left), unmake_tree(root.right))

    root = make_tree(ast.literal_eval(input()))

    val = int(input().strip())

    result_tree = insert_node(root, val)

```

```
    result = unmake_tree(result_tree)

    fptr.write(str(result) + '\n')

    fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	0	0.0419 sec	10.9 KB
Testcase 1	Easy	Hidden	Success	5	0.0484 sec	11.1 KB
Testcase 2	Easy	Hidden	Success	5	0.0473 sec	11.2 KB
Testcase 3	Easy	Hidden	Success	5	0.0434 sec	11 KB
Testcase 4	Easy	Hidden	Success	5	0.0438 sec	11.3 KB

No comments.

6. Sum the Nodes

Correct

Coding

Question description

Given the `root` of a binary tree, return the sum of all its nodes' values.

You may assume all values are integers.

Example Input Tree:
"""For example:
    1
     \
      2
     /
    3
"""
Input: root = 1
Expected Output: 6

Input: root = None
Output: 0

Example Input Tree
""" 1 """
Input: root = 1
Output: 1

Candidate's Solution Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast


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

#
# Complete the 'sum_tree' function below.
```

```
#
# The function is expected to return an INTEGER.
# The function accepts TreeNode root as parameter.
#

def sum_tree(root):
    # Write your code here
    if not root:
        return 0
    return root.val + sum_tree(root.left) + sum_tree(root.right)

if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    def make_tree(tree_tup):
        if tree_tup is None:
            return None # Base case
        elif len(tree_tup) != 3:
            print("Invalid input: ", tree_tup)
            return None # Invalid case
        # Happy case
        return TreeNode(tree_tup[0], make_tree(tree_tup[1]),
make_tree(tree_tup[2]))

    root = make_tree(ast.literal_eval(input()))

    result = sum_tree(root)

    fptr.write(str(result) + '\n')

    fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	0	0.0418 sec	10.9 KB
Testcase 1	Easy	Hidden	Success	5	0.0539 sec	10.9 KB

Testcase 2	Easy	Hidden	Success	5	0.0495 sec	11.2 KB
Testcase 3	Easy	Hidden	Success	5	0.0404 sec	10.9 KB
Testcase 4	Easy	Hidden	Success	5	0.0411 sec	11 KB

No comments.

7. Remove Node from a Binary Search Tree

Correct

Coding

Question description

Given the `root` of a binary search tree, remove the node with the value `val` into the tree. All nodes in the tree are guaranteed to be unique. Return the `root`.
If you need to replace a parent node with two children, use the **in-order successor** of that node, such that `replacement.val` is the smallest value **greater** than `removed.val`

```
# Example Input Tree:
#      7
#     / \
#    3  10
#       / \
#      8  12
# Input: root = 7, val = 10
# Expected Output: root = 7
# Expected output tree:
#      7
#     / \
#    3   12
#       /
#      8
```

```
#
# Note: In-order successor of 10 is 12
```

Candidate's Solution Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast


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

#
# Complete the 'remove_node' function below.
#
# The function is expected to return a TreeNode.
# The function accepts following parameters:
#  1. TreeNode root
#  2. INTEGER value
#

def remove_node(root, value):
    # Write your code here
    if not root:
        return root

    if value < root.val:
        root.left = remove_node(root.left, value)
    elif value > root.val:
        root.right = remove_node(root.right, value)
    else:
        if not root.left:
            return root.right
        elif not root.right:
```

```
            return root.left

        temp = find_min(root.right)
        root.val = temp.val
        root.right = remove_node(root.right, temp.val)

    return root

def find_min(node):
    current = node
    while current.left:
        current = current.left
    return current

if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    def make_tree(tree_tup):
        if tree_tup is None:
            return None
        elif len(tree_tup) != 3:
            print("Invalid input: ", tree_tup)
        else:
            return TreeNode(tree_tup[0], make_tree(tree_tup[1]),
make_tree(tree_tup[2]))

    def unmake_tree(root):
        if root is None:
            return None
        return (root.val, unmake_tree(root.left), unmake_tree(root.right))

    root = make_tree(ast.literal_eval(input()))

    value = int(input().strip())

    result_node = remove_node(root, value)

    result = unmake_tree(result_node)

    fptr.write(str(result) + '\n')

    fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	0	0.0442 sec	11.3 KB
Testcase 1	Easy	Hidden	Success	5	0.0539 sec	11 KB
Testcase 2	Easy	Hidden	Success	5	0.0466 sec	11 KB
Testcase 3	Easy	Hidden	Success	5	0.0462 sec	11.1 KB
Testcase 4	Easy	Hidden	Success	5	0.0443 sec	11 KB

No comments.