100

Rajasekhar Reddy Kolagotla

Other

PDF generated at: 30 Jul 2025 05:29:50 UTC

View this report on HackerRank

Score

100% • 80 / 80

scored in TIP102: Unit 8 Version B (Standard) - Summer 2025 in 21 min 6 sec on 29 Jul 2025 22:06:50 PDT

Candidate Information

Email	rajasekhar1131997@gmail.com
Test	TIP102: Unit 8 Version B (Standard) - Summer 2025
Candidate Packet	View
Taken on	29 Jul 2025 22:06:50 PDT
Time taken	21 min 6 sec/ 90 min
Personal Member ID	126663
Email Address with CodePath	rajasekhar1131997@gmail.com
Github username with CodePath	Rajasekhar1131997
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

Coding Questions • 60 / 60

Status	No.	Question	Time Taken	Skill	Score	Code Quality
✓	1	Sum of All Nodes in a Binary Tree Coding	3 min 30 sec	-	20/20	-

✓	2	Remove Node in a BST Coding	7 min 57 sec	-	20/20	-
✓	3	Balanced BST Coding	4 min 47 sec	-	20/20	-

Multiple Choice + Debugging • 20 / 20

Status	No.	Question	Time Taken	Skill	Score	Code Quality
✓	4	What is the time complexity of mystery_function()? Multiple Choice	40 sec	-	5/5	-
✓	5	Given the following code, which of the following best represents the values of the tree with root? Multiple Choice	43 sec	-	5/5	-
✓	6	Given the following binary tree, which of the following arrays best represents its postorder traversal? Multiple Choice	1 min 20 sec	-	5/5	-
✓	7	Debugging Coding	1 min 42 sec	-	5/5	-

1. Sum of All Nodes in a Binary Tree

Correct

Coding

Question description

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

Example 1:
Input: root = [1, 2, 3, 4, 5]
Output: 15

Example 2:
Input: root = [1, 2, 3, None, 4, None, 5]
Output: 15

Example 3:
Input: root = []
Output: 0

Candidate's Solution

Language used: Python 3

```python
#!/bin/python3

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

from collections import deque

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

```

```
19  def sum_of_nodes(root):
20      # Write your code here
21      if not root:
22          return 0
23      left_subtree = sum_of_nodes(root.left)
24      right_subtree = sum_of_nodes(root.right)
25
26      return root.val + left_subtree + right_subtree
27
28  def list_to_tree(nodes):
29      if not nodes:
30          return None
31
32      root = TreeNode(nodes[0])
33      queue = deque([root])
34      i = 1
35
36      while i < len(nodes):
37          current = queue.popleft()
38
39          if nodes[i] is not None:
40              current.left = TreeNode(nodes[i])
41              queue.append(current.left)
42          i += 1
43
44          if i < len(nodes) and nodes[i] is not None:
45              current.right = TreeNode(nodes[i])
46              queue.append(current.right)
47          i += 1
48
49      return root
50
51  if __name__ == '__main__':
52      outfile = open(os.environ['OUTPUT_PATH'], 'w')
53      input_data = sys.stdin.read().strip()
54
55      input_data = input_data.splitlines()
56
57      for data in input_data:
58          if data.strip() == "":
59              continue
60
61          data = data.replace('null', 'None')
62          tree_list = ast.literal_eval(data)
63
64          root = list_to_tree(tree_list)
```

```
65          result = sum_of_nodes(root)
66          outfile.write(str(result) + '\n')
67      outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Basic Tree	Easy	Hidden	Success	0	0.0346 sec	11 KB
Tree with Single Node	Easy	Hidden	Success	0	0.029 sec	10.9 KB
Full Binary Tree	Easy	Hidden	Success	0	0.0284 sec	11 KB
Tree with Missing Nodes (Incomplete)	Easy	Hidden	Success	0	0.0358 sec	10.8 KB
All Nodes are None	Easy	Hidden	Success	0	0.0341 sec	11 KB
Tree with Negative Values	Easy	Hidden	Success	0	0.0364 sec	10.6 KB
Tree with Only Left Children	Easy	Hidden	Success	0	0.028 sec	11 KB
Tree with Only Right Children	Easy	Hidden	Success	0	0.0432 sec	10.9 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0284 sec	11 KB

No comments.

2. Remove Node in a BST

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` of the modified tree.

If you need to replace a parent node with two children, use the in-order successor of that node. The in-order successor is the node with the smallest value greater than the value of the removed node.

Example 1:
Input: root = [5, 3, 6, 2, 4, None, 7], key = 3
Output: [5, 4, 6, 2, None, None, 7]

Example 2:
Input: root = [5, 3, 6, 2, 4, None, 7], key = 0
Output: [5, 3, 6, 2, 4, None, 7]

Example 3:
Input: root = [], key = 0
Output: []

Candidate's Solution

Language used: Python 3

```
#!/bin/python3

import math
import os
import random
import re
```

```
import sys
import ast
import json

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

def delete_node(root, key):
    # Write your code here
    if not root:
        return None
    if key < root.val:
        root.left = delete_node(root.left, key)
    elif key > root.val:
        root.right = delete_node(root.right, key)
    else:
        if not root.left:
            return root.right
        elif not root.right:
            return root.left
        successor = find_successor(root.right)
        root.val = successor.val
        root.right = delete_node(root.right, successor.val)
    return root

def find_successor(node):
    if not node:
        return None
    current = node
    while current.left:
        current = current.left
    return current


def build_tree_from_list(lst):
    if not lst:
        return None

    nodes = [TreeNode(val) if val is not None else None for val in lst]
    kids = nodes[::-1]
    root = kids.pop()

    for node in nodes:
```

```
53          if node:
54              if kids: node.left = kids.pop()
55              if kids: node.right = kids.pop()
56
57      return root
58
59  def tree_to_list(root):
60      if not root:
61          return []
62
63      result, queue = [], [root]
64      while any(queue):
65          node = queue.pop(0)
66          if node:
67              result.append(node.val)
68              queue.append(node.left)
69              queue.append(node.right)
70          else:
71              result.append(None)
72
73      # Remove trailing None values
74      while result and result[-1] is None:
75          result.pop()
76
77      return result
78
79  if __name__ == '__main__':
80      input_data = sys.stdin.read().strip().splitlines()
81
82      for data in input_data:
83          tree_data, val = data.split('],')
84          tree_data += ']'
85          val = int(val.strip())
86
87          tree_list = ast.literal_eval(tree_data)
88
89          root = build_tree_from_list(tree_list)
90          result = delete_node(root, val)
91
92          print(tree_to_list(result))
93
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED

Basic Case	Easy	Hidden	Success	0	0.0319 sec	11.1 KB
Delete Leaf Node	Easy	Hidden	Success	0	0.0308 sec	11.1 KB
Delete Node with One Child	Easy	Hidden	Success	0	0.039 sec	11.1 KB
Delete Node with Two Children	Easy	Hidden	Success	0	0.0306 sec	11.1 KB
Delete Root Node	Easy	Hidden	Success	0	0.0395 sec	11.1 KB
Delete Node Not Present in Tree	Easy	Hidden	Success	0	0.0357 sec	11.1 KB
Delete Node in an Empty Tree	Easy	Hidden	Success	0	0.0333 sec	11 KB
Delete Only Node in Tree	Easy	Hidden	Success	0	0.0365 sec	11.1 KB
Tree with All Nodes Having Same Value	Easy	Hidden	Success	0	0.0394 sec	11.1 KB
Delete Node with Deep Children	Easy	Hidden	Success	0	0.0292 sec	11.1 KB

Left-Skewed Tree (Only Left Children)	Easy	Hidden	Success	0	0.0324 sec	11.1 KB
Right-Skewed Tree (Only Right Children)	Easy	Hidden	Success	0	0.0378 sec	11.1 KB
Balanced Tree	Easy	Hidden	Success	0	0.0296 sec	11 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0336 sec	11.1 KB

No comments.

3. Balanced BST

Correct

Coding

Question description

Given the `root` of a binary tree, write a function that returns `True` if the tree is height-balanced, and `False` otherwise.
A binary tree is height-balanced if the depth of the two subtrees of every node never differs by more than one.

Example 1:
Input: root = [3, 9, 20, None, None, 15, 7]
Output: True

Example 2:
Input: root = [1, 2, 2, 3, 3, None, None, 4, 4]
Output: False

Example 3:
Input: root = []
Output: True

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


def is_balanced(root):
    # Write your code here
    def check(node):
        if not node:
            return True, 0
        left_balanced, left_height = check(node.left)
        right_balanced, right_height = check(node.right)
        balanced = (left_balanced and right_balanced and abs(left_height-right_height) <=1)
        return balanced, max(left_height, right_height) + 1
    result, _ = check(root)
    return result


def list_to_tree(lst):
    if not lst:
        return None
    root = TreeNode(lst[0])
    queue = [root]
    i = 1
```

```
    while i < len(lst):
        current = queue.pop(0)
        if i < len(lst) and lst[i] is not None:
            current.left = TreeNode(lst[i])
            queue.append(current.left)
        i += 1
        if i < len(lst) and lst[i] is not None:
            current.right = TreeNode(lst[i])
            queue.append(current.right)
        i += 1
    return root

if __name__ == '__main__':
    outfile = open(os.environ['OUTPUT_PATH'], 'w')
    input_data = sys.stdin.read().strip()

    input_data = input_data.splitlines()

    for data in input_data:
        if data.strip() == "":
            continue

        data = data.replace('null', 'None')
        tree_list = ast.literal_eval(data)

        root = list_to_tree(tree_list)
        result = is_balanced(root)
        outfile.write(str(result) + '\n')
    outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Basic Case: True	Easy	Hidden	Success	0	0.0268 sec	11 KB
Basic Case: False	Easy	Hidden	Success	0	0.0298 sec	10.9 KB
Empty Tree	Easy	Hidden	Success	0	0.033 sec	11 KB

Simple Balanced Tree	Easy	Hidden	Success	0	0.0249 sec	11 KB
Single Node Tree	Easy	Hidden	Success	0	0.028 sec	10.9 KB
Completely Unbalanced Tree (Skewed to the Right)	Easy	Hidden	Success	0	0.0334 sec	11 KB
Completely Unbalanced Tree (Skewed to the Left)	Easy	Hidden	Success	0	0.0292 sec	10.9 KB
Large Balanced Tree	Easy	Hidden	Success	0	0.0314 sec	11 KB
Tree with Missing Nodes (Balanced)	Easy	Hidden	Success	0	0.0384 sec	10.9 KB
Tree with Only Left Subtree	Easy	Hidden	Success	0	0.0292 sec	11 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.028 sec	10.9 KB

No comments.

4. What is the time complexity of mystery_function()?

Correct

Multiple Choice

Question description

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

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

    left_result = mystery_function(node.left_child)
    right_result = mystery_function(node.right_child)

    return left_result + right_result + 1
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ O(1)
- ○ O(log n)
- ● **O(n)** ✓
- ○ O(n log n)

No comments.

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

Correct

Multiple Choice

Question description

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

a = TreeNode('a')
b = TreeNode('b')
c = TreeNode('c')
d = TreeNode('d')
e = TreeNode('e')

a.left = b
a.right = c
b.left = d
b.right = e

root = a
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [x]   a
       / \
     b   c
    / \
  d   e

- [ ]   a
       / \
     d   e
    / \
  b   c

- [ ]   a
       / \
     e   d
    / \
  c   b

- [ ]   a
       / \
     c   b
    / \
  e   d

No comments.

6. Given the following binary tree, which of the following arrays best represents its postorder traversal?

Correct

Multiple Choice

Question description

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

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [ ] [1, 2, 3, 4, 5]
- [ ] [4, 2, 5, 1, 3]
- [x] **[4, 5, 2, 3, 1]**
- [ ] [1, 4, 2, 5, 3]

No comments.

7. Debugging

Correct

Coding

Question description

The following code incorrectly implements `is_valid_bst()` . Implemented correctly, `is_valid_bst()` accepts the `root` of a binary tree and returns `True` if the tree is a valid binary search tree and `False` otherwise.

Identify any bug(s) within the given implementation and correct the code so that it successfully passes the provided test cases.

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

# Helper function to create a tree from a list (level-order traversal)
def create_tree(values):
    if not values:
        return None
    nodes = [TreeNode(val) if val is not None else None for val in values]
    kids = nodes[::-1]
    root = kids.pop()
    for node in nodes:
        if node:
            if kids:
                node.left = kids.pop()
            if kids:
                node.right = kids.pop()
    return root



# Complete the `is_valid_bst` function below
def is_valid_bst(root):
    def validate(node, low, high):
        if not node:
            return True
        if node.val <= low or node.val >= high:
            return False
        return (validate(node.left, low, node.val) and
                validate(node.right, node.val, high))  # Error here

    return validate(root, float('-inf'), float('inf'))
if __name__ == "__main__":
    import sys
    input_data = sys.stdin.read().strip().split("\n")
```

```
    results = []

    for line in input_data:
        values = eval(line)  # Parse the input as a list
        root = create_tree(values)
        results.append(is_valid_bst(root))

    for res in results:
        print(res)

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	5	0.0283 sec	10.8 KB

No comments.