100 Venkata Naga Sri Sai Pranavi Kolipaka

Other

PDF generated at: 31 Jul 2025 20:23:44 UTC

View this report on HackerRank

Score

100% • 80 / 80

scored in TIP102: Unit 9 Version A (Standard) - Summer 2025 in 44 min 29 sec on 31 Jul 2025 12:37:34 PDT

Candidate Information

Email	kolipakavnssaipranavi@gmail.com
Test	TIP102: Unit 9 Version A (Standard) - Summer 2025
Candidate Packet	View
Taken on	31 Jul 2025 12:37:34 PDT
Time taken	44 min 29 sec/ 90 min
Personal Member ID	129054
Email Address with CodePath	kolipakavnssaipranavi@gmail.com
Github username with CodePath	Pranavi2002
Invited by	CodePath

Suspicious Activity detected

Code similarity

Code similarity • 2 questions

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	Level Order Traversal Coding	14 min 55 sec	-	20/20	-
✓	2	Right View of Binary Tree Coding	9 min 6 sec	-	20/20 ⚐	-
✓	3	Construct Tree from Preorder Array Coding	6 min 51 sec	-	20/20 ⚐	-

Multiple Choice + Debugging • 20 / 20

Status	No.	Question	Time Taken	Skill	Score	Code Quality
✓	4	What is the time complexity of find_max_depth()? Multiple Choice	3 min 28 sec	-	5/5	-
✓	5	Which of the following options most accurately creates the tree depicted below? Multiple Choice	1 min 3 sec	-	5/5	-
✓	6	What is the value of output? Multiple Choice	6 min 27 sec	-	5/5	-

✓	7	Debug this code! Coding	2 min 22 sec	-	5/5	-

1. Level Order Traversal

✓ Correct

Coding

Question description

Given the `root` of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

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

Example 2:
Input: root = [1]
      1
Output: [[1]]

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

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


def level_order_traversal(root):
    # Write your code here
    if not root:
        return []

    from collections import deque
    visited = []
    stored_nodes= deque([root])

    while stored_nodes:
        level = []
        level_size = len(stored_nodes)

        for _ in range(level_size):
            node = stored_nodes.popleft()
            level.append(node.val)

            if node.left:
                stored_nodes.append(node.left)
            if node.right:
                stored_nodes.append(node.right)

        visited.append(level)

    return visited
def build_tree(nodes):
    if not nodes:
        return None

    root = TreeNode(nodes[0])
    queue = deque([root])
    i = 1

    while queue and i < len(nodes):
        current = queue.popleft()

        if nodes[i] is not None:
            current.left = TreeNode(nodes[i])
            queue.append(current.left)
        i += 1

        if i < len(nodes) and nodes[i] is not None:
            current.right = TreeNode(nodes[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 = build_tree(tree_list)
        result = level_order_traversal(root)
        outfile.write(str(result) + '\n')
    outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Basic Case	Easy	Hidden	Success	0	0.0287 sec	11 KB
Single Node Tree	Easy	Hidden	Success	0	0.0302 sec	11 KB

Empty Tree	Easy	Hidden	Success	0	0.0303 sec	10.9 KB
Left-Skewed Tree	Easy	Hidden	Success	0	0.0285 sec	11 KB
Right-Skewed Tree	Easy	Hidden	Success	0	0.0317 sec	11 KB
Complete Binary Tree	Easy	Hidden	Success	0	0.0309 sec	11 KB
Sparse Tree	Easy	Hidden	Success	0	0.0307 sec	11 KB
Tree with Missing Nodes at Different Levels	Easy	Hidden	Success	0	0.0288 sec	11 KB
Tree with All None Values	Easy	Hidden	Success	0	0.031 sec	10.9 KB
Larger Tree	Easy	Hidden	Success	0	0.0296 sec	11 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0306 sec	11 KB

No comments.

2. Right View of Binary Tree

Correct

Coding

Question description

Given the `root` of a binary tree, imagine yourself standing on the right side of it. Return a list of the values of the nodes you can see, ordered from top to bottom.

```
Example 1:
Input: root = [1,2,3, None, 5, None,4]
     1
    / \
   2   3
    \   \
     5   4
Output: [1, 3, 4]

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

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

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

```

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


def right_view(root):
    # Write your code here
    result = []

    def dfs(node, depth):
        if not node:
            return

        # If we are visiting this depth for the first time
        if depth == len(result):
            result.append(node.val)

        # Visit right child first to get the right view
        dfs(node.right, depth + 1)
        #  the right child at a level might be missing, but there could be a left child that is visible from the right side because nothing is blocking it
        dfs(node.left, depth + 1)

    dfs(root, 0)
    return result
def list_to_tree(lst):
    if not lst:
        return None

    root = TreeNode(lst[0])
    queue = deque([root])
    i = 1

    while i < len(lst):
        node = queue.popleft()
        if lst[i] is not None:
            node.left = TreeNode(lst[i])
            queue.append(node.left)
        i += 1
        if i < len(lst) and lst[i] is not None:
            node.right = TreeNode(lst[i])
            queue.append(node.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 = right_view(root)
        outfile.write(str(result) + '\n')
    outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Single Node Tree	Easy	Hidden	Success	0	0.0309 sec	10.9 KB
Tree with Only Left Children	Easy	Hidden	Success	0	0.029 sec	11 KB
Tree with Only Right Children	Easy	Hidden	Success	0	0.0294 sec	10.8 KB
Full Binary Tree with Depth 3:	Easy	Hidden	Success	0	0.0343 sec	11 KB
Unbalanced Tree with Varying Levels:	Easy	Hidden	Success	0	0.0293 sec	11 KB

Tree with Multiple Nodes but Sparse Right Children	Easy	Hidden	Success	0	0.0308 sec	11 KB
Tree with All Nodes Having Only Right Children Except One Left Child	Easy	Hidden	Success	0	0.0284 sec	11 KB
Tree with One Node per Level	Easy	Hidden	Success	0	0.0305 sec	10.8 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0313 sec	11 KB

No comments.

3. Construct Tree from Preorder Array

Correct

Coding

Question description

Given an array of unique integers `preorder`, which represents the **preorder traversal** of a binary search tree, construct the tree and return its root.

It is **guaranteed** that a binary search tree can be constructed from the given array.

A **binary search tree** is a binary tree where for every node, any descendant of `Node.left` has a value **strictly less than** `Node.val`, and any descendant of `Node.right` has a value **strictly greater than** `Node.val`.

A **preorder traversal** of a binary tree displays the value of the node first, then traverses `Node.left`, then traverses `Node.right`.

```
Example 1:
Input: preorder = [8, 5, 1, 7, 10, 12]
Output: [8, 5, 10, 1, 7, None, 12]
Explanation:
The tree structure is:
     8
    / \
   5   10
  / \    \
 1   7   12

Example 2:
Input: preorder = [4, 2]
Output: [4, 2]
Explanation:
The tree structure is:
   4
  /
 2

Example 3:
Input: preorder = [1]
Output: [1]
Explanation:
The tree structure is:
   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

def insert_node(root, val):
    if val < root.val:
        if root.left:
            insert_node(root.left, val)
        else:
            root.left = TreeNode(val)
    else:
        if root.right:
            insert_node(root.right, val)
        else:
            root.right = TreeNode(val)

def bst_from_preorder(preorder):
    # Write your code here
    index = 0 # to keep track of current root in preorder

    def helper(lower=float('-inf'), upper=float('inf')):
        nonlocal index
        if index == len(preorder):
            return None

        val = preorder[index]

        #if current val is out of bst range, return None
        if val < lower or val > upper:
            return None

        # construct node and recurse
        index += 1
        root = TreeNode(val)
        root.left = helper(lower, val)
        root.right = helper(val, upper)
        return root

    return helper()
def print_tree(root):
    """ Helper function to print the tree nodes in level order. """
    if not root:
        return []
    queue = [root]
    result = []
    while queue:
        current = queue.pop(0)
        if current:
            result.append(current.val)
            queue.append(current.left)
            queue.append(current.right)
        else:
            result.append("None")
    # Remove trailing "None" values that represent missing nodes at the end of the tree
    while result and result[-1] == "None":
        result.pop()
    return result


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

        tree_list = ast.literal_eval(data)

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

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Basic Case	Easy	Hidden	Success	0	0.0273 sec	11 KB
[4, 2]	Easy	Hidden	Success	0	0.0285 sec	11 KB

Single Node	Easy	Hidden	Success	0	0.0305 sec	11 KB
All Elements Forming a Right Skewed Tree	Easy	Hidden	Success	0	0.0278 sec	11 KB
All Elements Forming a Left Skewed Tree	Easy	Hidden	Success	0	0.0308 sec	11 KB
Complex Case with Multiple Levels	Easy	Hidden	Success	0	0.028 sec	11 KB
Empty Input	Easy	Hidden	Success	0	0.0317 sec	11 KB
Two Elements with Larger First Element	Easy	Hidden	Success	0	0.0369 sec	11 KB
Two Elements with Smaller First Element	Easy	Hidden	Success	0	0.03 sec	11 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.028 sec	11 KB

No comments.

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

✓ Correct

Multiple Choice

Question description

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

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

    left_depth = find_max_depth(root.left)
    right_depth = find_max_depth(root.right)

    return max(left_depth, right_depth) + 1
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

○ O(n * log n)

● O(n) ✓

○ O(log n)

○ O(n^2)

No comments.

5. Which of the following options most accurately creates the tree depicted below? ✓ Correct

Multiple Choice

Question description

Given the following class TreeNode which of the following options most accurately creates the tree depicted below?

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

#       10
#      /  \
#     5    15
#    / \     \
#   2   7    20
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

○ <pre> <code>root = TreeNode(10) root.left(TreeNode(5)) root.right(TreeNode(15)) root.left.left(TreeNode(2)) root.left.right(TreeNode(7)) root.right.right(TreeNode(20)) </code></pre> <p>&nbsp;</p>

● <pre> <code>root = TreeNode(10) root.left = TreeNode(5) root.right = TreeNode(15) root.left.left = TreeNode(2) root.left.right = TreeNode(7) root.right.right = TreeNode(20) </code></pre> <p>&nbsp;</p> ✓

○ <pre> <code>root = TreeNode(10) root.left = TreeNode(5) root.right = TreeNode(15) left.left = TreeNode(2) left.left = TreeNode(7) right.right = TreeNode(20) </code></pre> <p>&nbsp;</p>

○ <pre> <code>root = TreeNode(10) root.left = TreeNode(5) root.right = TreeNode(15) TreeNode(5).left = TreeNode(2) TreeNode(5).right = TreeNode(7) TreeNode(15).right = TreeNode(20) </code></pre> <p>&nbsp;</p>

ⓘ No comments.

6. What is the value of output?

✓ Correct

Multiple Choice

Question description

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

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

def helper(node):
    if not node:
        return 0
    return 1 + helper(node.left) + helper(node.right)

def mystery_function(root):
    if not root:
        return "empty"

    left_count = count_nodes(root.left)
    right_count = count_nodes(root.right)


    if left_count > right_count:
        return "left"
    elif right_count > left_count:
        return "right"
    else:
        return "equal"

root = TreeNode(1)
root.left = TreeNode(2)
root.left.left = TreeNode(4)
root.right = TreeNode(3)
root.right.left = TreeNode(5)
root.right.right = TreeNode(6)

output = mystery_function(root)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

○ &quot;empty&quot;

○ &quot;left&quot;

● &quot;right&quot; ✓

○ &quot;equal&quot;

ⓘ No comments.

7. Debug this code!

✓ Correct

Coding

Question description

The provided code incorrectly implements `is_valid_bst()`. When correctly implemented, `is_valid_bst()` should accept the `root` of a tree and return `True` if the tree is a valid binary search tree (BST), and `False` otherwise.

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with values **less than** the node's key
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.
- The tree may not have duplicate values

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**

```
#!/bin/python3

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

```

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


def is_valid_bst(root):
    def validate(node, low=float('-inf'), high=float('inf')):
        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))

    return validate(root)
def build_tree(nodes):
    if not nodes:
        return None

    root = TreeNode(nodes[0])
    queue = [root]
    index = 1

    while queue and index < len(nodes):
        node = queue.pop(0)

        if nodes[index] is not None:
            node.left = TreeNode(nodes[index])
            queue.append(node.left)
        index += 1

        if index < len(nodes) and nodes[index] is not None:
            node.right = TreeNode(nodes[index])
            queue.append(node.right)
        index += 1

    return root


if __name__ == '__main__':
    input_data = sys.stdin.read().strip()
    input_data = input_data.replace('null', 'None')

    nodes = ast.literal_eval(input_data)
    root = build_tree(nodes)

    result = is_valid_bst(root)
    print(result)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Pass/Fail Case	Easy	Hidden	Success	5	0.0319 sec	11 KB

ⓘ No comments.