94.6

Muhammad Tanveer

Software Engineer

PDF generated at: 18 Jul 2024 03:03:20 UTC

View this report on HackerRank

Score

94.6% • 175 / 185

scored in CodePath TIP103: Unit 5 Assessment - Summer 2024 in 67 min 23 sec on 17 Jul 2024 18:54:09 PDT

Candidate Information

Email	tanveerm176@gmail.com
Test	CodePath TIP103: Unit 5 Assessment - Summer 2024
Candidate Packet	View
Taken on	17 Jul 2024 18:54:09 PDT
Time taken	67 min 23 sec/ 90 min
Personal Email Address	tanveerm176@gmail.com
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

Binary Trees	100%	Binary Search Trees	100%

Questions

Status	No.	Question	Time Taken	Skill	Score
✓	1	What is the postorder traversal of this binary tree? Multiple Choice	3 min 15 sec	-	5/5
✓	2	Binary Tree Traversal Multiple Choice	44 sec	-	5/5
✗	3	Inorder Traversal of a complete binary tree Multiple Choice	1 min 22 sec	-	0/5
✓	4	Binary Search Tree Efficiency Multiple Choice	2 min 3 sec	-	5/5

✓	5	Get the sum of all left leaf nodes in a binary tree Multiple Choice	3 min 36 sec	-	5/5
✗	6	Validate if a binary tree is actually a binary search tree Multiple Choice	12 min 28 sec	-	0/5
✓	7	Average of Levels in Binary Tree Coding	6 min 29 sec	-	50/50
✓	8	Serialize and Deserialize Binary Tree Coding	21 min 11 sec	-	50/50
✓	9	Binary Search Tree Iterator Coding	6 min 29 sec	-	50/50
✓	10	What is the missing piece of code found in the push function? Multiple Choice	9 min 36 sec	-	5/5

1. What is the postorder traversal of this binary tree?

Correct

Multiple Choice

Question description

What is the postorder traversal of this binary tree?

```
  A
 / \
```

```
  B   C
 /   / \
D   E   F
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [ ] A, B, C, D, E, F
- [x] **D, B, E, F, C, A** ✓
- [ ] D, B, A, C, E, F
- [ ] D, E, F, B, C, A

No comments.

2. Binary Tree Traversal

Correct

Multiple Choice

Question description

Suppose we want to make a copy of a binary tree, what type of traversal would be best suited for this problem?

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [x] Preorder
- [ ] Postorder
- [ ] Inorder

No comments.

---

3. Inorder Traversal of a complete binary tree

Incorrect

Multiple Choice

Question description

Given that the inorder traversal of a complete binary tree is: 3, 7, 2, 5, 9, 1, 10 what is the BFS traversal of the same tree?

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [ ] 3, 7, 2, 5, 9, 1, 10
- [x] 1, 2, 3, 5, 7, 9, 10

- ○ 5, 2, 9, 3, 7, 1, 10
- ○ 5, 7, 1, 3, 2, 9, 10 ✓
- ○ 5, 1, 7, 3, 2, 9, 10

No comments.

4. Binary Search Tree Efficiency

Correct

Multiple Choice

Question description

Knowing that the tree below is a binary search tree, which numbers would we pass through while looking for 22 with the most efficient method?

Candidate's Solution

Options: (Expected answer indicated with a tick)

- 10
- **10, 20** ✓
- 10, 9, 20
- 10, 9, 20, 15

No comments.

5. Get the sum of all left leaf nodes in a binary tree

Correct

Multiple Choice

Question description

The following code is meant to get the sum of all left leaf nodes in a binary tree. Given the code and the tree below, what will the code output?

Java:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {

    public int sumOfLeftLeaves(TreeNode root) {
```

```
        return helper(root, 0);
    }

    public int helper(TreeNode root, int sumSoFar) {
        if (root == null) {
            return sumSoFar;
        }

        if (root.left != null) {
            sumSoFar += root.left.val;
        }

        return helper(root.left, sumSoFar) + helper(root.right, sumSoFar);
    }
}
```

Python:

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

def sumOfLeftLeaves(root):
    def helper(root, sumSoFar):
        if not root:
            return sumSoFar

        if root.left:
            sumSoFar += root.left.val

        return helper(root.left, sumSoFar) + helper(root.right,sumSoFar)

    return helper(root, 0)
```

```
  3
 / \
9   20
   /  \
 15    7
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ 9
- ● 24 ✓
- ○ 31
- ○ 51
- ○ 114

No comments.

6. Validate if a binary tree is actually a binary search tree

Incorrect

Multiple Choice

Question description

The following code is meant to validate if a binary tree is actually a binary search tree. Will we produce the right output with the following code? If not, which lines need to be amended?

Java:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValidBST(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }

    public boolean isValidBST(TreeNode root, int min, int max) {
        if (root == null) {
            return true;
        }
        if (root.val >= max || root.val <= min) {
            return false;
        }
        return isValidBST(root.left, Math.min(min, root.val), Math.max(min, root.val)) &&
            isValidBST(root.right, Math.min(min, root.val), Math.max(min, root.val));
    }
}
```

Python:

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

def isValidBST(root):
    def helper(root, min, max):
        if not root:
            return True

        if root.val >= max or root.val <= min:
```

```
20            return False
21
22        return helper(root.left, min(min, root.val), max(min, root.val)) and
23            helper(root.right, min(min, root.val), max(min, root.val))
24
25    return helper(root, -float("inf"), float("inf"))
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ It will produce the right code
- ● **Line 19 needs to be fixed**
- ○ Line 22 needs to be fixed
- ○ Line 23 needs to be fixed ✓
- ○ Lines 22 and 23 needs to be fixed
- ○ More than 2 lines need to be fixed

No comments.

7. Average of Levels in Binary Tree

Correct

Coding Binary Trees

Question description

Given a non-empty binary tree, return the average value of the nodes on each level in the form of a list.

Example:

Input:
```
   3
  / \
 9  20
   /  \
  15   7
```

Output: [3, 14.5, 11]

Explanation:
The average value of nodes on level 0 is 3, on level 1 is 14.5, and on level 2 is 11. Hence return [3, 14.5, 11].

Candidate's Solution

Language used: **Python 3**

```python
#!/usr/bin/env python

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

def input_binary_tree():
    input_values = input().split()
    index = 0
    num_nodes = int(input_values[index])
    index += 1
    if (num_nodes == 0):
        return None

    nodes = []
    current_parent_index = 0
```

```
18
19      root = TreeNode(int(input_values[index]))
20      index += 1
21      nodes.append(root)
22
23      for i in range(1, num_nodes, 2):
24          left_val = int(input_values[index])
25          index += 1
26          if (left_val != -1):
27              left = TreeNode(left_val)
28              nodes.append(left)
29              nodes[current_parent_index].left = left
30
31          right_val = int(input_values[index])
32          index += 1
33          if (right_val != -1):
34              right = TreeNode(right_val)
35              nodes.append(right)
36              nodes[current_parent_index].right = right
37
38          current_parent_index += 1
39
40      return root
41  """
42  class TreeNode:
43      def __init__(self, x):
44          self.val = x
45          self.left = self.right = None
46  """
47
48  from collections import deque
49  def average_of_levels(root):
50      """
51      Write your code here
52      :type root: TreeNode
53      :rtype: List[double]
54      """
55
56      # if the tree is empty return an empty list
57      if not root:
58          return []
59
60      result = []
61      queue = deque([root])
62
63      while queue:
```

```
64          level_sum = 0
65          level_count = len(queue)
66
67          # process all nodes at the current level
68          for _ in range(level_count):
69              node = queue.popleft()
70
71              level_sum += node.val
72
73              if node.left:
74                  queue.append(node.left)
75
76              if node.right:
77                  queue.append(node.right)
78
79          result.append(level_sum/level_count)
80
81      return result
82
83  root = input_binary_tree()
84  averages = average_of_levels(root)
85
86  for average in averages:
87      print(round(average,1))
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample	Success	20	0.0241 sec	9.37 KB
Testcase 2	Easy	Hidden	Success	10	0.0308 sec	9.5 KB
Testcase 3	Easy	Hidden	Success	5	0.0431 sec	9.45 KB
Testcase 4	Easy	Hidden	Success	5	0.0335 sec	9.38 KB

Testcase 5	Easy	Hidden	Success	10	0.0252 sec	9.44 KB

No comments.

8. Serialize and Deserialize Binary Tree

Correct

Coding | Binary Trees | Binary Search Trees

Question description

Design an algorithm to serialize and deserialize a binary tree.

In this problem, we want to ensure that if we serialize a binary tree into a string, the string can be deserialized back to the original tree

For example, you may serialize the following tree

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

into a string seen as: `"[1,2,3,*,*,4,5]"` , with * representing a null node.

You don't need to follow the format seen above, so feel free to serialize the tree in a way that makes sense to you.

Candidate's Solution

Language used: **Python 3**

```python
#!/usr/bin/env python

class TreeNode:
    def __init__(self, x):
        self.val = x
```

```
6           self.left = self.right = None
7
8
9   def insert(val, root):
10      if val < root.val:
11          if root.left is None:
12              root.left = TreeNode(val)
13          else:
14              insert(val, root.left)
15      else:
16          if root.right is None:
17              root.right = TreeNode(val)
18          else:
19              insert(val, root.right)
20
21
22  def input_bst():
23      input_values = map(int, input().split())
24      num_nodes = next(input_values)
25      if num_nodes == 0:
26          return None
27
28      root = TreeNode(next(input_values))
29
30      for i in range(1, num_nodes):
31          insert(next(input_values), root)
32
33      return root
34
35
36  def description(root):
37      if root is None:
38          return " "
39
40      queue = []
41
42      output = str(root.val)
43      queue.append(root)
44      cursor = 0
45
46      while cursor < len(queue):
47          node = queue[cursor]
48          cursor += 1
49
50          if node.left is not None:
51              output += " " + str(node.left.val)
```

```
52              queue.append(node.left)
53
54          if node.right is not None:
55              output += " " + str(node.right.val)
56              queue.append(node.right)
57
58      return output
59
60
61  from collections import deque
62
63  """
64  class TreeNode:
65      def __init__(self, x):
66          self.val = x
67          self.left = self.right = None
68  """
69
70  def serialize(root):
71      """Encodes a tree to a single string.
72
73      :type root: TreeNode
74      :rtype: str
75      """
76      def serialize_helper(node):
77          if node is None:
78              return result.append('None')
79
80          else:
81              result.append(str(node.val))
82              serialize_helper(node.left)
83              serialize_helper(node.right)
84
85      result = []
86      serialize_helper(root)
87      return ','.join(result)
88
89
90
91
92  def deserialize(data):
93      """Decodes your encoded data to tree.
94
95      :type data: str
96      :rtype: TreeNode
97      """
```

```
98
99      def deserialize_helper():
100         if data[0] == 'None':
101             data.popleft()
102             return None
103
104         node = TreeNode(int(data.popleft()))
105         node.left = deserialize_helper()
106         node.right = deserialize_helper()
107         return node
108
109     data = deque(data.split(','))
110     return deserialize_helper()
111
112
113
114
115
116
117
118 root = input_bst()
119 new_root = deserialize(serialize(root))
120 print(description(new_root))
121
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	20	0.0263 sec	9.36 KB
Testcase 1	Easy	Hidden	Success	5	0.0288 sec	9.43 KB
Testcase 2	Easy	Hidden	Success	5	0.0287 sec	9.36 KB
Testcase 3	Easy	Hidden	Success	5	0.0429 sec	9.46 KB

Testcase 4	Easy	Hidden	Success	5	0.0293 sec	9.32 KB
Testcase 5	Easy	Hidden	Success	10	0.033 sec	9.29 KB

No comments.

9. Binary Search Tree Iterator

Correct

Coding | Binary Trees | Binary Search Trees

Question description

Implement an iterator over a binary search tree. Your iterator will be initialized with the root node of a BST.
Calling `next()` should return the next smallest number in the BST.

The common iterator methods `next()` and `hasNext()` should run in average O(1) time and uses O(h) memory, where h is the height of the tree.

Candidate's Solution

Language used: **Python 3**

```
#!/usr/bin/env python

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


def insert(val, root):
    if val < root.val:
        if root.left is None:
            root.left = TreeNode(val)
        else:
```

```
14              insert(val, root.left)
15      else:
16          if root.right is None:
17              root.right = TreeNode(val)
18          else:
19              insert(val, root.right)
20
21
22  def input_bst():
23      input_values = map(int, input().split())
24      num_nodes = next(input_values)
25      if num_nodes == 0:
26          return None
27
28      root = TreeNode(next(input_values))
29
30      for i in range(1, num_nodes):
31          insert(next(input_values), root)
32
33      return root
34
35
36  """
37  class TreeNode:
38      def __init__(self, x):
39          self.val = x
40          self.left = self.right = None
41  """
42  class BSTIterator(object):
43      def __init__(self, root):
44          """
45          :type root: TreeNode
46          """
47          self.stack = []
48          self.leftmost_inorder(root)
49
50      def leftmost_inorder(self, root: TreeNode):
51          # Helper fucntion to push all the left most nodes of the tree to the
    stack
52          while root:
53              self.stack.append(root)
54              root = root.left
55
56
57      def has_next(self):
58          """
```

```
59          :rtype: bool
60          """
61          return len(self.stack) > 0
62
63
64      def next(self):
65          """
66          :rtype: int
67          """
68          topmost_node = self.stack.pop()
69          if topmost_node.right:
70              self.leftmost_inorder(topmost_node.right)
71
72          return topmost_node.val
73  root = input_bst()
74  iter = BSTIterator(root)
75  while iter.has_next():
76      print(iter.next())
77
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	20	0.0266 sec	9.28 KB
Testcase 1	Easy	Hidden	Success	10	0.0319 sec	9.25 KB
Testcase 2	Easy	Hidden	Success	5	0.0248 sec	9.13 KB
Testcase 3	Easy	Hidden	Success	5	0.0247 sec	9.2 KB
Testcase 4	Easy	Hidden	Success	5	0.025 sec	9.11 KB

Testcase 5	Easy	Hidden	Success	5	0.0262 sec	9.23 KB

No comments.

10. What is the missing piece of code found in the push function?

Correct

Multiple Choice

Question description

Given two queues, implement a last-in-first-out (LIFO) stack. The implemented stack supports all the functions of a normal stack (`push` , `top` , `pop` , and `empty`).
The `MyStack` class consists of the following functions:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a queue, which means that only `push to back` , `peek/pop from front` , `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:

Input
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2

```
myStack.pop(); // return 2
myStack.empty(); // return False
```

What is the missing piece of code found in the `push` function?

```
class MyStack:

    def __init__(self):
        self.q = deque()
        self.t = None

    def push(self, x: int) -> None:
        if self.t:
            // add missing line of code here

        self.t = x

    def pop(self) -> int:
        result = self.t

        newq = deque()
        while len(self.q) > 1:
            newq.append(self.q.popleft())

        self.t = self.q.popleft() if self.q else None
        self.q = newq

        return result

    def top(self) -> int:
        return self.t

    def empty(self) -> bool:
        return self.t is None
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- self.q = deque()
- self.q.append(self.q.popleft())
- **self.q.append(self.t)** ✓
- return self.l[-1]
- none of the above

No comments.