100

ryan

Other

PDF generated at: 21 Jul 2024 21:33:59 UTC

View this report on HackerRank

Score

100% • 80 / 80

scored in CodePath TIP101: Unit 6 Assessment, Version A - Summer 2024 in 74 min 7 sec on 21 Jul 2024 13:18:22 PDT

Candidate Information

Email	concepting@protonmail.com
Test	CodePath TIP101: Unit 6 Assessment, Version A - Summer 2024
Candidate Packet	View
Taken on	21 Jul 2024 13:18:22 PDT
Time taken	74 min 7 sec/ 90 min
Work Experience	< 1 years
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

Status	No.	Question	Time Taken	Skill	Score
✓	1	Tree Structure Multiple Choice	1 min 41 sec	-	5/5
✓	2	Extract Even Values from a Singly Linked List Multiple Choice	1 min 38 sec	-	5/5

✓	3	Time Complexity Multiple Choice	11 min 15 sec	-	5/5
✓	4	Space Complexity Multiple Choice	33 sec	-	5/5
✓	5	Find Nth From End (LL) Coding	11 min 36 sec	-	20/20 ⚐
✓	6	Shuffle (LL) Coding	6 min 58 sec	-	20/20
✓	7	Find Intersection (LL) Coding	40 min 18 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> <p><br /> <!-- notionvc: 335ce73b-fcbe-45c0-9041-d7f7f5efe34b --><!-- notionvc: 43cffeaa-2d37-4866-90ce-a0371fad32e6 --></p>

● **<pre> <code class="language-python"> a / \ / \ x y / \ \ e m p</code></pre> <p><br /> <!-- notionvc: 328084e6-4df2-4ae9-834b-5e008b6dce37 --></p>** ✓

○ <pre> <code class="language-python"> a / \ / \ x y / \ \ m e p</code></pre> <p><br /> <!-- notionvc: a3c29946-ba02-4e14-9e8b-f5aa1d00f0eb --><!-- notionvc: ad2c3bdc-8b34-4881-996a-e480c9a25a6a --><!-- notionvc: 169fa24e-3d44-491e-a51c-1156fbd4cbc7 --><!-- notionvc: 10453584-991a-4f8b-a411-722a914c99cb --></p>

○ <pre> <code class="language-python"> a / \ / \ x y / \ / e m p </code></pre> <p><br /> <!-- 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. Extract Even Values from a Singly Linked List

✓ Correct

Multiple Choice

Question description

What is the value of `output`?
Note: If `output` is a `Node`, the answer will include all nodes in the linked list represented by `output`.

```
class Node:
    def __init__(self, value, next_node=None):
        self.value = value
        self.next = next_node

def mystery_function(head):
    current = head
    output = []

    while current:
        if current.value % 2 == 0:
            output.append(current.value)
        current = current.next

    return output
```

```
# Input List: 1 -> 2 -> 3 -> 4
head = Node(1, Node(2, Node(3, Node(4))))

output = mystery_function(head)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [● <p><code>[2, 4]</code></p>] ✓
- ○ <p><code>2 -&gt; 4</code></p>
- ○ <p><code>[1, 3]</code></p>
- ○ <p><code>1 -&gt; 3</code></p>

No comments.

3. Time Complexity

✓ Correct

Multiple Choice

Question description

What is the time complexity of the following code? Assume `n` is the length of the linked list.

```
def print_linked_list(head):
    current = head
    while current:
        print(current.value)
        current = current.next
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- O(1)
- [✓ O(n)] ✓
- O(n^2)
- O(n^3)

No comments.

4. Space Complexity

✓ Correct

Multiple Choice

Question description

What is the space complexity of the following code? Assume `n` is the length of the linked list.

```
def print_linked_list(head):
    current = head
    while current:
        print(current.value)
        current = current.next
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [✓ O(1)] ✓
- O(n)
- O(n²)
- O(n³)

No comments.

5. Find Nth From End (LL)

Correct

Coding

Question description

Given a linked list, write a function **find_nth_from_end** that finds the nth node from the end of the list and returns its value.

```
# Example Input:
  2
  a->b->c->d->e->f

# Output:
  e
```

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys



#
# Complete the 'find_nth_from_end' function below.
#
# The function is expected to return a STRING.
# The function accepts following parameters:
#  1. HEAD head
#  2. INTEGER n
#

class Node:
   def __init__(self, value, next_node = None):
       self.value = value
       self.next = next_node

def find_nth_from_end(head, n):
    # Write your code here
    # using two pointer approach
    point_one = head
    point_two = head

    for i in range(n):
        if point_one is None:
            return None
        point_one = point_one.next

    while point_one is not None:
        point_one = point_one.next
        point_two = point_two.next

    if point_two is not None:
        return point_two.value
    return None

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

    # Helper function to convert str -> linked list
    def str_to_linked_list(vals_str):
        if vals_str == "None":
            return None
        vals = vals_str.split("->")
        temp_head = Node("temp")
        temp_curr = temp_head
        for val in vals:
            temp_curr.next = Node(val.strip())
            temp_curr = temp_curr.next
        return temp_head.next #Don't keep the temp head

    # Helper function to convert linked list to str
    def linked_list_to_str(head):
        list_str = ""
        curr = head
        while curr:
            list_str += curr.value
            if curr.next:
                list_str += "->"
            curr = curr.next
        if len(list_str) == 0:
            return "None"
        return list_str

    # Read and convert test input
    #inp = input()
    n = int(input())
    head = str_to_linked_list(input())

    # Call the function
    answer = find_nth_from_end(head, n)

    # Turn the list into a string
    list_str = linked_list_to_str(head)

    # Bundle result in format <answer>|<linked list>
    result = str(answer) + "\n" + list_str

    fptr.write(result)
    fptr.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	0	0.0338 sec	10.3 KB
Testcase 1	Easy	Hidden	Success	0	0.0318 sec	10.4 KB
Testcase 2	Easy	Hidden	Success	0	0.0373 sec	10.3 KB
Testcase 3	Easy	Hidden	Success	0	0.0329 sec	10.4 KB
Testcase 4	Easy	Hidden	Success	20	0.0383 sec	10.4 KB

No comments.

6. Shuffle (LL)

Correct

Coding

Question description

Write a function that shuffles a linked list by swapping adjacent items. If there are an odd number of elements, the tail should not change position.

```
# Example 1:
# Input: a->b->c->d->e->f
# Output: b->a->d->c->f->e

# Notice that adjacent items have swapped position:
#  a swapped with b
#  c swapped with d
#  e swapped with f
```

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys



#
# Complete the 'shuffle' function below.
#
# The function is expected to return nothing.
# The function accepts HEAD head as parameter.
#

class Node:
   def __init__(self, value, next_node = None):
       self.value = value
       self.next = next_node

def shuffle(head):
    # Write your code here
    current = head

    while current and current.next:
        current.value, current.next.value = current.next.value, current.value
        current = current.next.next






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

    # Helper function to convert str -> linked list
    def str_to_linked_list(vals_str):
        if vals_str == "None":
            return None
        vals = vals_str.split("->")
        temp_head = Node("temp")
        temp_curr = temp_head
        for val in vals:
            temp_curr.next = Node(val.strip())
            temp_curr = temp_curr.next
        return temp_head.next #Don't keep the temp head

    # Helper function to convert linked list to str
    def linked_list_to_str(head):
        list_str = ""
        curr = head
        while curr:
            list_str += curr.value
            if curr.next:
                list_str += "->"
            curr = curr.next
        if len(list_str) == 0:
            return "None"
        return list_str

    # Read and convert test input
    head = str_to_linked_list(input())

    # Call the function
    answer = shuffle(head)

    # Turn the list back into a string
    list_str = linked_list_to_str(head)

    fptr.write(list_str)
    fptr.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	0	0.0364 sec	10.4 KB
Testcase 1	Easy	Hidden	Success	0	0.0393 sec	10.4 KB
Testcase 2	Easy	Hidden	Success	0	0.0353 sec	10.4 KB
Testcase 3	Easy	Hidden	Success	20	0.0398 sec	10.4 KB
Testcase 4	Easy	Hidden	Success	0	0.0382 sec	10.3 KB

No comments.

7. Find Intersection (LL)

Correct

Coding

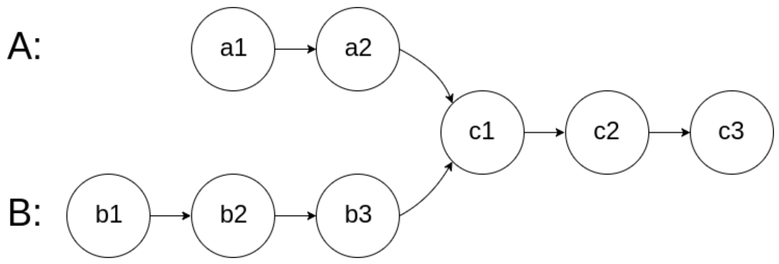
Question description

Given the heads of two singly linked lists, return the node at which the two lists intersect. If the two linked lists do not intersect, return None. You may not modify either of the linked lists.

EXAMPLE

list_a is   a1->a2->c1->c2->c3
list_b is b1->b2->b3->c1->c2->c3

intersection point is c1



Example image from: https://leetcode.com/problems/intersection-of-two-linked-lists/description/

Candidate's Solution

Language used: **Python 3**

```
#!/bin/python3

import math
import os
import random
import re
import sys



#
# Complete the 'find_intersection' function below.
#
# The function is expected to return a NODE.
# The function accepts following parameters:
#  1. HEAD head_a
#  2. HEAD head_b
#

class Node:
   def __init__(self, value, next_node = None):
       self.value = value
       self.next = next_node

def get_len(head):
    length = 0
    current = head
    while current:
        length += 1
        current = current.next
    return length

def find_intersection(head_a, head_b):
    # Write your code here
    length_a = get_len(head_a)
    length_b = get_len(head_b)

    if length_a > length_b:
        for i in range(length_a - length_b):
            head_a = head_a.next
    else:
        for i in range(length_b - length_a):
            head_b = head_b.next

    while head_a and head_b:
        if head_a == head_b:
            return head_a
        head_a = head_a.next
        head_b = head_b.next
    return None


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

    # Helper function to convert str -> linked list
    def str_to_linked_list_without_repetition(vals_str, nodes_dict={}):
        if vals_str == "None":
            return None, {}
        vals = [x.strip() for x in vals_str.split("->")]
        temp_head = Node("temp")
        temp_curr = temp_head
        for val in vals:
            if val in nodes_dict:
                temp_curr.next = nodes_dict[val]
            else:
                temp_curr.next = Node(val)
                nodes_dict[val] = temp_curr.next
            temp_curr = temp_curr.next
        return temp_head.next, nodes_dict #Don't keep the temp head

    # Helper function to convert linked list to str
    def linked_list_to_str(head):
        list_str = ""
        curr = head
        while curr:
            list_str += curr.value
            if curr.next:
                list_str += "->"
            curr = curr.next
        if len(list_str) == 0:
            return "None"
        return list_str

    # Read and convert test input
    head_a, dict_a = str_to_linked_list_without_repetition(input())
    head_b, _ = str_to_linked_list_without_repetition(input(), dict_a)

    # Call the function
    answer = find_intersection(head_a, head_b)
    if answer is not None:
        answer = answer.value

    # Turn the lists into a string
    list_str_a = linked_list_to_str(head_a)
    list_str_b = linked_list_to_str(head_b)

    # Bundle result in format <answer>\n<original linked list>
    result = str(answer) + "\n" + list_str_a + "\n" + list_str_b

    fptr.write(result)
    fptr.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.0478 sec	10.4 KB
Testcase 1	Easy	Hidden	Success	0	0.0365 sec	10.3 KB

Testcase 2	Easy	Hidden	Success	0	0.0319 sec	10.3 KB
Testcase 3	Easy	Hidden	Success	20	0.032 sec	10.4 KB
Testcase 4	Easy	Hidden	Success	0	0.0356 sec	10.5 KB
Testcase 5	Easy	Hidden	Success	0	0.0355 sec	10.5 KB

No comments.