100

Rajasekhar Reddy Kolagotla

Other

PDF generated at: 7 Jul 2025 23:03:09 UTC

View this report on HackerRank

Score

100% • 80 / 80

scored in TIP102: Unit 5 Version A (Standard) - Summer 2025 in 29 min 35 sec on 7 Jul 2025 15:31:51 PDT

Candidate Information

Email	rajasekhar1131997@gmail.com
Test	TIP102: Unit 5 Version A (Standard) - Summer 2025
Candidate Packet	View
Taken on	7 Jul 2025 15:31:51 PDT
Time taken	29 min 35 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 • 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	Create a Linked List Coding	6 min 18 sec	-	20/20	-

✓	2	Insert Node into List Coding	5 min 19 sec	-	20/20 ⚑	-
✓	3	More Prime Numbers Coding	7 min 9 sec	-	20/20 ⚑	-

Multiple Choice + Debugging • 20 / 20

Status	No.	Question	Time Taken	Skill	Score	Code Quality
✓	4	What is the output of the following code snippet? Multiple Choice	3 min 58 sec	-	5/5	-
✓	5	Time Complexity Multiple Choice	1 min 13 sec	-	5/5	-
✓	6	What is the output of the following code snippet? Multiple Choice	2 min 16 sec	-	5/5	-
✓	7	Find the bug Coding	3 min 4 sec	-	5/5	-

1. Create a Linked List

✓ Correct

Coding

Question description

Write a function `create_linked_list(values)` that creates a linked list from a list of numbers, `nums` and returns the `head` of the linked list.

```
Example 1:

Input: nums = [1, 2, 3, 4, 5]
Output: 1 -> 2 -> 3 -> 4 -> 5 -> None


Example 2:
Input: nums = []
Output: None
```

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python

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

class Node:
    def __init__(self, node_data):
        self.data = node_data
        self.next = None

def print_linked_list(head):
    current = head
    while current:
        if current.next:
            sys.stdout.write(str(current.data) + " -> ")
        else:
            sys.stdout.write(str(current.data) + "\n")
        current = current.next
def create_linked_list(values):
    if not values:
        return None
    head = Node(values[0])
    current = head
    for val in values[1:]:
        current.next = Node(val)
        current = current.next
    return head


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

    def ll_to_str(head):
        list_str = ""
        curr = head
        while curr:
            list_str += str(curr.data)
            if curr.next:
                list_str += "->"
            curr = curr.next
        if len(list_str) == 0:
            return "None"
        return list_str

    test_str = input()
    while(test_str != "END"):
        # Convert input string to list of param strings
        param_list = ast.literal_eval(test_str)

        # TODO: Edit function name and prepare result string
        result_raw = create_linked_list(param_list)
        result = ll_to_str(result_raw)

        # Write output and check for another test case
        outfile.write(str(result) + '\n')
        test_str = input()

    outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
List with Multiple Elements	Easy	Hidden	Success	0	0.0392 sec	11 KB

Empty Linked List	Easy	Hidden	Success	0	0.0428 sec	11 KB
Single Element List	Easy	Hidden	Success	0	0.0364 sec	11 KB
List with Duplicate Elements	Easy	Hidden	Success	0	0.0318 sec	10.9 KB
List with Special Characters	Easy	Hidden	Success	0	0.0313 sec	10.9 KB
List with Nested Lists	Easy	Hidden	Success	0	0.0412 sec	10.9 KB
Mixed Data Types	Easy	Hidden	Success	0	0.0285 sec	11 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0326 sec	10.9 KB

No comments.

2. Insert Node into List

Correct

Coding

Question description

You are given a class `ListNode` representing a node in a singly linked list, and a function `print_linked_list` to print the linked list.

Your task is to implement the function `insert` that inserts a new node with a given value at a specified position in the linked list.

```
Example 1: Insert in the middle
Input:
Linked list: 1 -> 2 -> 4
Insert value: 3
Position: 2

Output:
1 -> 2 -> 3 -> 4 -> None


Example 2: Insert in the middle of a two-node list
Input:
Linked list: 10 -> 20
Insert value: 15
Position: 1

Output:
10 -> 15 -> 20 -> None
```

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python

import math
import os
import random
import re
import sys

class ListNode:
    def __init__(self, node_data):
        self.data = node_data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def insert_node(self, node_data):
        node = ListNode(node_data)

        if not self.head:
            self.head = node
        else:
            self.tail.next = node

        self.tail = node

def print_linked_list(node, sep, fptr):
    while node:
        fptr.write(str(node.data))

        node = node.next

        if node:
            fptr.write(sep)


#
# Complete the 'insert' function below.
#
# The function is expected to return an INTEGER_LINKED_LIST.
# The function accepts following parameters:
#  1. INTEGER_LINKED_LIST head
#  2. INTEGER value
#  3. INTEGER position
#

#
# For your reference:
#
# ListNode:
#     int data
#     ListNode next
#
#

def insert(head, value, position):
    # Write your code here
    new_node = ListNode(value)
    if position == 0:
        new_node.next = head
        return new_node
    current = head
    index = 0
    while current and index < position - 1:
        current = current.next
        index += 1
    if not current:
        return head

    new_node.next = current.next
    current.next = new_node

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

    input_line = input().strip()
    while(input_line != "END"):

        inputs = input_line.split(',')

        head = LinkedList()
        if len(inputs) == 3:
            try:
                head_count_str = inputs[0].strip()
                if head_count_str.startswith('ListNode(') and head_count_str.endswith(')'):
                    head_value = int(re.search(r'\d+', head_count_str).group())
                    head.insert_node(head_value)  # Insert initial head node
                else:
                    head_count = int(head_count_str) if head_count_str != 'None' else 0

                value = int(inputs[1].strip())
                position = int(inputs[2].strip())
            except (ValueError, IndexError):
                print("Invalid input format. Please provide a valid head_count, value, and position.")
                fptr.write('Invalid input format. Please provide a valid head_count, value, and position.\n')
                fptr.close()
                sys.exit(1)
        else:
            print("Invalid input format. Please provide exactly three values.")
```

```
                fptr.write('Invalid input format. Please provide exactly three
    values.\n')
                fptr.close()
                sys.exit(1)

            result = insert(head.head, value, position)
            print_linked_list(result, ' -> ', fptr)
            fptr.write('\n')
            input_line = input().strip()

        fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Insert into an Empty List	Easy	Hidden	Success	0	0.0249 sec	10.4 KB
Insert at the Beginning of a Non-Empty List	Easy	Hidden	Success	0	0.0246 sec	10.4 KB
Insert at the End of a Non-Empty List	Easy	Hidden	Success	0	0.024 sec	10.3 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0239 sec	10.4 KB

No comments.

3. More Prime Numbers

Correct

Coding

Question description

You are given the heads of two singly linked lists, `head_a` and `head_b`. Your task is to determine which list contains more **prime numbers**. The function should return the head of the list that has the greater count of prime numbers.
If both lists have the same number of prime numbers, return `head_a`.

Constraints

- The lists contain at least one node and at most `10^3` nodes.
- Node values are integers in the range `[-10^5, 10^5]`.
- The `is_prime(n)` function is provided and can be used to determine if a number is prime.

A prime number is defined as a natural number greater than 1 that has no positive divisors other than 1 and itself.

```
Example 1:
List A: 2 -> 3 -> 4
a1 = SinglyLinkedListNode(2)
a2 = SinglyLinkedListNode(3)
a3 = SinglyLinkedListNode(4)
a1.next = a2
a2.next = a3

List B: 5 -> 6 -> 8
b1 = SinglyLinkedListNode(5)
b2 = SinglyLinkedListNode(6)
b3 = SinglyLinkedListNode(8)
b1.next = b2
b2.next = b3

Output: 2 (head of List A, because List A has two primes: [2,3] while List B has one: [5])


Example 2:
List A: 7 -> 8 -> 9
a1 = SinglyLinkedListNode(7)
a2 = SinglyLinkedListNode(8)
a3 = SinglyLinkedListNode(9)
a1.next = a2
a2.next = a3

List B: 11 -> 12 -> 13
b1 = SinglyLinkedListNode(11)
```

b2 = SinglyLinkedListNode(12)
b3 = SinglyLinkedListNode(13)
b1.next = b2
b2.next = b3

Output: 7 (head of List A, because both have the same number of primes: [7] vs [11, 13] but we retu rn head_a by default)

Candidate's Solution

Language used: **Python 3**

```python
import math
import os
import random
import re
import sys
import ast

class SinglyLinkedListNode:
    def __init__(self, node_data):
        self.data = node_data
        self.next = None

class SinglyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def insert_node(self, node_data):
        node = SinglyLinkedListNode(node_data)
        if not self.head:
            self.head = node
        else:
            self.tail.next = node
        self.tail = node

# Helper function to create a linked list from a list of values
def create_linked_list(vals):
    temp = SinglyLinkedListNode(0)  # Dummy node
    current = temp
    for val in vals:
        current.next = SinglyLinkedListNode(val)
        current = current.next
    return temp.next
```

```python

# Function to check if a number is prime
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True


# Complete the 'most_primes_list' function below.
#
# The function is expected to return a SinglyLinkedListNode.
# The function accepts two SinglyLinkedListNode parameters: head_a and
head_b.
#
def most_primes_list(head_a, head_b):
    count_a = 0
    current = head_a
    while current:
        if is_prime(current.data):
            count_a += 1
        current = current.next
    count_b = 0
    current = head_b
    while current:
        if is_prime(current.data):
            count_b += 1
        current = current.next
    if count_a == count_b:
        return head_a
    if count_a >count_b:
        return head_a
    else:
        return head_b

import sys

# Helper function to print linked list
```

```
def print_linked_list(node, sep, fptr):
    while node:
        fptr.write(str(node.data))
        node = node.next
        if node:
            fptr.write(sep)

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

    try:
        input_data = sys.stdin.read().strip().split("\n")  # Read all input
at once
    except EOFError:
        input_data = []

    for line in input_data:
        if not line.strip():
            continue

        input_list = ast.literal_eval(line)  # Parse input as list of lists

        head_a = create_linked_list(input_list[0])
        head_b = create_linked_list(input_list[1])

        result = most_primes_list(head_a, head_b)
        print_linked_list(result, ' -> ', fptr)
        fptr.write('\n')

    fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.0299 sec	11 KB
Testcase 1	Easy	Hidden	Success	0	0.0279 sec	11 KB

Testcase 2	Easy	Hidden	Success	0	0.0288 sec	11 KB
Testcase 3	Easy	Hidden	Success	0	0.0357 sec	11 KB
Testcase 4	Easy	Hidden	Success	0	0.028 sec	11 KB
Testcase 5	Easy	Hidden	Success	0	0.0365 sec	11 KB
Testcase 6	Easy	Hidden	Success	0	0.0276 sec	11 KB
Testcase 7	Easy	Hidden	Success	0	0.0304 sec	11 KB
Testcase 8	Easy	Hidden	Success	0	0.0282 sec	11 KB
Pass/Fail Testcases	Easy	Hidden	Success	20	0.0381 sec	11 KB

No comments.

4. What is the output of the following code snippet?

✓ Correct

Multiple Choice

Question description

```
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed
        self.pets_received = 0

    def receive_pet(self):
        self.pets_received += 1
        return f"{self.name} has received a pet!"

    def bark(self):
        return f"{self.name} says woof!"

# Create Dog objects
dog1 = Dog("Buddy", "Poodle")
dog2 = Dog("Bella", "Labrador")

# Dog interactions
print(dog2.bark())
print(dog1.receive_pet())
print(dog1.receive_pet())
print(dog1.pets_received)
print(dog2.receive_pet())
print(dog2.pets_received)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [● ✓] <pre> <code>Bella says woof! Buddy has received a pet! Buddy has received a pet! 2 Bella has received a pet! 1</code></pre> <p>&nbsp;</p>
- ○ <pre> <code>Buddy says woof! Buddy has received a pet! Buddy has received a pet! 2 Bella has received a pet! 1</code></pre> <p>&nbsp;</p>

○ <pre> <code>Bella says woof! Buddy has received a pet! Buddy has received a pet! 1 Bella has received a pet! 2</code></pre> <p>&nbsp;</p>

○ <pre> <code>Bella says woof! Buddy has received a pet! Bella has received a pet! 1 Bella has received a pet! 1</code></pre> <p>&nbsp;</p>

No comments.

5. Time Complexity

✓ Correct

Multiple Choice

Question description

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

```
# Definition for singly-linked list.
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def mystery_function(head):
    if not head or not head.next:
        return None

    current = head
    while current.next and current.next.next:
        current = current.next
    current.next = None

    return head
```

```
# Example Usage:
head = ListNode('a')
head.next = ListNode('b')
head.next.next = ListNode('c')
head.next.next.next = ListNode('d')
new_head = mystery_function(head) # Expected Output: a -> b -> c
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- O(1)
- O(log n)
- [✓ O(n)] ✓
- O(n^2)

No comments.

6. What is the output of the following code snippet?

✓ Correct

Multiple Choice

Question description

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def mystery_function(head):
    nums1 = []
    nums2 = []
    current = head
    while current:
        if current.val % 2 == 0:
            nums1.append(str(current.val))
        else:
            nums2.append(str(current.val))
        current = current.next
    return " -> ".join(nums1 + nums2)

# Create Linked List: 1 -> 2 -> 3 -> 4 -> 5
head = ListNode(1)
head.next = ListNode(2)
head.next.next = ListNode(3)
head.next.next.next = ListNode(4)
head.next.next.next.next = ListNode(5)

print(mystery_function(head))
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [✓ ● 2 -&gt; 4 -&gt; 1 -&gt; 3 -&gt; 5] ✓
- ○ 1 -&gt; 3 -&gt; 5 -&gt; 2 -&gt; 4

○ 1 -&gt; 2 -&gt; 3 -&gt; 4 -&gt; 5

○ 5 -&gt; 4 -&gt; 3 -&gt; 2 -&gt; 1

No comments.

7. Find the bug

Correct

Coding

Question description

The provided code incorrectly implements `count_nodes_with_value()`. When implemented correctly, `count_nodes_with_value()` accepts the head of a singly linked list and a value, and returns the number of nodes in the linked list with value `val`.

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/python

import math
import os
import random
import re
import sys

class SinglyLinkedListNode:
    def __init__(self, node_data):
        self.data = node_data
        self.next = None

class SinglyLinkedList:
```

```python
    def __init__(self):
        self.head = None
        self.tail = None

    def insert_node(self, node_data):
        node = SinglyLinkedListNode(node_data)

        if not self.head:
            self.head = node
        else:
            self.tail.next = node

        self.tail = node

def print_singly_linked_list(node, sep, fptr):
    while node:
        fptr.write(str(node.data))

        node = node.next

        if node:
            fptr.write(sep)


#
# Complete the 'count_nodes_with_value' function below.
#
# The function is expected to return an INTEGER.
# The function accepts following parameters:
#  1. INTEGER_SINGLY_LINKED_LIST head
#  2. INTEGER val
#

#
# For your reference:
#
# SinglyLinkedListNode:
#     int data
#     SinglyLinkedListNode next
#
#

def count_nodes_with_value(head, val):
    count = 0
    current = head

    while current:
        if current.data == val:
            count += 1
        current = current.next

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

    input_data = input().strip()
    while(input_data != "END"):
        if input_data == "None":
            fptr.write(str(0))
            fptr.write('\n')
            input_data = input().strip()
        else:
            list_part, value_part = input_data.split(', ')

            values = list(map(int, list_part.split(' -> ')))

            value = int(value_part)

            head = None
            tail = None

            for head_item in values:
                new_node = SinglyLinkedListNode(head_item)
                if head is None:
                    head = new_node
                    tail = head
                else:
                    tail.next = new_node
                    tail = new_node

            result = count_nodes_with_value(head, value)
            fptr.write(str(result))
            fptr.write('\n')
            input_data = input().strip()
    fptr.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED

Testcase 0	Easy	Hidden	Success	0	0.0226 sec	10.1 KB
a head that doesn't exist	Easy	Hidden	Success	0	0.0274 sec	10.1 KB
Pass/Fail Case	Easy	Hidden	Success	5	0.0312 sec	10.1 KB

ⓘ No comments.