Alex Kaiser

BASIC INFO

Email: alex-kaiser@lambdastudents.com
Test: Lambda School - Whiteboard Fitness Assessment I
Solved: 3/3
Score: 175/175
Finished On: 29 Jul 2020
Time Taken: 138m/3h

Labels: -

Task	Solve Time	Score	Similarity
remove_kth_from_end	75min	50/50	low
first_not_repeating_character	14min	50/50	none
uncover_spy	48min	75/75	none

Task details: **remove_kth_from_end**

Description:

Write a function that receives as input the head node of a linked list and an integer k. Your function should remove the kth node from the end of the linked list and return the head node of the updated list.

For example, if we have the following linked list:
`(20) -> (19) -> (18) -> (17) -> (16) -> (15) -> (14) -> (13) -> (12) -> (11) -> null`

The head node would refer to the node `(20)`. Let k = 4, so our function should remove the 4th node from the end of the linked list, the node `(14)`.

After the function executes, the state of the linked list should be:
`(20) -> (19) -> (18) -> (17) -> (16) -> (15) -> (13) -> (12) -> (11) -> null`

If k is longer than the length of the linked list, the linked list should not be changed.

Can you implement a solution that performs a single pass through the linked list and doesn't use any extra space?

Note: When reading the tests, the linked list contents are enumerated in between square brackets; this does NOT mean the inputs are arrays.

For example, a test input of `head: [2, 4 ,6]` indicates that the input is a singly-linked list
`(2) -> (4) -> (6) -> null` whose head is the first element in the linked list.

Solution (main.py3):

```
# Singly-linked lists are already defined with this interface:
# class ListNode(object):
#   def __init__(self, x):
#     self.value = x
#     self.next = None
#

def length(head):
    if head is None:
        return 0
    else:
       return 1 + length(head.next)

def remove_kth_from_end(head, k):

    slow = head
    fast = head
```

```
if k == 0 or k > length(head):
    return head

for r in range(0, k):
    fast = fast.next

    if fast == None:
        head = head.next
        return head

while fast.next  != None:
    fast = fast.next
    slow = slow.next

slow.next = slow.next.next


return head
```

Task details: first_not_repeating_character

Description:

Given a string `s` consisting of small English letters, find and return the first instance of a non-repeating character in it. If there is no such character, return `'_'`.

Example

- For `s = "abacabad"`, the output should be `first_not_repeating_character(s) = 'c'`.

  There are 2 non-repeating characters in the string: `'c'` and `'d'`. Return `c` since it appears in the string first.

- For `s = "abacabaabacaba"`, the output should be `first_not_repeating_character(s) = '_'`.

  There are no characters in this string that do not repeat.

Solution (main.py3):

```
from collections import Counter

def first_not_repeating_character(s):

    # using counter from Collections
    count = Counter(s)

    for c in s:
        if count[c] == 1:
            return c

    return '_'
```

Task details: uncover_spy

Description:

In a city-state of `n` people, there is a rumor going around that one of the `n` people is a spy for the neighboring city-state.

The spy, if it exists:

1. Does not trust anyone else.
2. Is trusted by everyone else (he's good at his job).
3. Works alone; there are no other spies in the city-state.

You are given a list of pairs, `trust`. Each `trust[i] = [a, b]` represents the fact that person `a` trusts person `b`.

If the spy exists and can be found, return their identifier. Otherwise, return `-1`.

Example 1:

```
Input: n = 2, trust = [[1, 2]]
Output: 2
Explanation: Person 1 trusts Person 2, but Person 2 does not trust Person 1, so Person 2
```

Example 2:

```
Input: n = 3, trust = [[1, 3], [2, 3]]
Output: 3
Explanation: Person 1 trusts Person 3, and Person 2 trusts Person 3, but Person 3 does no
```

Example 3:

```
Input: n = 3, trust = [[1, 3], [2, 3], [3, 1]]
Output: -1
Explanation: Person 1 trusts Person 3, Person 2 trusts Person 3, and Person 3 trusts Pers
```

Example 4:

```
Input: n = 3, trust = [[1, 2], [2, 3]]
Output: -1
Explanation: Person 1 trusts Person 2, and Person 2 trusts Person 3. However, in this si
```

Example 5:

```
Input: n = 4, trust = [[1, 3],[1, 4],[2, 3],[2, 4],[4, 3]]
Output: 3
Explanation: Person 1 trusts Person 3 and Person 4, Person 2 trusts Person 3 and Person 4
```

Solution (main.py3):

```
from collections import defaultdict
```

```
def uncover_spy(n, trust):

    count = defaultdict(int)

    # if (n + 1) > len(trust):
    #     return -1

    if n == 1:
        return 1

    for t in trust:
        count[t[0]] -= 1
        count[t[1]] += 1

    for i, j in count.items():
        if j == (n - 1):
            return i

    return -1
```