Even-After-Odd-Nodes

May 14, 2020

0.0.1 Problem Statement

Given a linked list with integer data, arrange the elements in such a manner that all nodes with even numbers are placed after odd numbers. **Do not create any new nodes and avoid using any other data structure. The relative order of even and odd elements must not change.**

Example: * `linked list = 1 2 3 4 5 6` * `output = 1 3 5 2 4 6`

```
In [34]: class Node:
             def __init__(self, data):
                 self.data = data
                 self.next = None
```

0.0.2 Exercise - Write the function definition here

```
In [2]: def even_after_odd(head):
            """
            :param - head - head of linked list
            return - updated list with all even elements are odd elements
            """
            pass
```

Hide Solution

```
In [ ]: """
        parameter: - head of the given linked list
        return: - head of the updated list with all even elements placed after odd elements
        """
        #--------------------------------------------------#
        '''
        The Idea: Traverse the given LinkedList, and build two sub-lists: EVEN and ODD.
        For this purpose, we will use four helper references, that denotes starting and
        current ending of EVEN and ODD sub-list respectively.

        1. For each Node in the LinkedList, check if its data is even/odd.
        Change the "next" reference (pointer) of each Node, based on the following rules:
         - First even valued Node will be referenced by head of EVEN sub-list
         - Subsequent even valued Node will be appended to the tail of EVEN sub-list

         - First odd valued Node will be referenced by head of ODD sub-list
```

```
 - Subsequent odd valued Node will be appended to the tail of ODD sub-list

2. After the loop, append the EVEN sub-list to the tail of ODD sub-list.
'''
#--------------------------------------------------#
def even_after_odd(head):

    if head is None:
        return head

    # Helper references
    ''' `even_head` and `even_tail` represents the starting and current ending of the "E
    even_head = None
    even_tail = None

    ''' `odd_head` and `odd_tail` represents the starting and current ending of the "ODD
    odd_head = None
    odd_tail = None

    current = head                  # <-- "current" represents the current Node.

    # Loop untill there are Nodes available in the LinkedList
    while current:                  # <-- "current" will be updated at the end of each i

        next_node = current.next    # <-- "next_node" represents the next Node w.r.t. th

        if current.data % 2 == 0:   # <-- current Node is even

            # Below
            if even_head is None:   # <-- Make the current Node as the starting Node of
                even_head = current     # `even_head` will now point where `current` is
                even_tail = even_head
            else:                   # <-- Append the current even node to the tail of EV
                even_tail.next = current
                even_tail = even_tail.next
        else:
            if odd_head is None:    # <-- Make the current Node as the starting Node of
                odd_head = current
                odd_tail = odd_head
            else:                   # <-- Append the current odd node to the tail of ODD
                odd_tail.next = current
                odd_tail = odd_tail.next
        current.next = None
        current = next_node         # <-- Update "head" Node, for next iteration

    if odd_head is None:            # <-- Special case, when there are no odd Nodes
        return even_head
    odd_tail.next = even_head       # <-- Append the EVEN sub-list to the tail of ODD su

    return odd_head
```

0.0.3 Test - Let's test your function

```
In [35]: # helper functions for testing purpose
         def create_linked_list(arr):
             if len(arr)==0:
                 return None
             head = Node(arr[0])
             tail = head
             for data in arr[1:]:
                 tail.next = Node(data)
                 tail = tail.next
             return head

         def print_linked_list(head):
             while head:
                 print(head.data, end=' ')
                 head = head.next
             print()
```

```
In [36]: def test_function(test_case):
             head = test_case[0]
             solution = test_case[1]

             node_tracker = dict({})
             node_tracker['nodes'] = list()
             temp = head
             while temp:
                 node_tracker['nodes'].append(temp)
                 temp = temp.next

             head = even_after_odd(head)
             temp = head
             index = 0
             try:
                 while temp:
                     if temp.data != solution[index] or temp not in node_tracker['nodes']:
                         print("Fail")
                         return
                     temp = temp.next
                     index += 1
                 print("Pass")
             except Exception as e:
                 print("Fail")
```

```
In [40]: arr = [1, 2, 3, 4, 5, 6]
```

```
         solution = [1, 3, 5, 2, 4, 6]

         head = create_linked_list(arr)
         test_case = [head, solution]
         test_function(test_case)
```

```
Pass
```

```
In [39]: arr = [1, 3, 5, 7]
         solution = [1, 3, 5, 7]

         head = create_linked_list(arr)
         test_case = [head, solution]
         test_function(test_case)
```

```
Pass
```

```
In [38]: arr = [2, 4, 6, 8]
         solution = [2, 4, 6, 8]
         head = create_linked_list(arr)
         test_case = [head, solution]
         test_function(test_case)
```

```
Pass
```