Solution Review: Problem Challenge 2

We'll cover the following

Rearrange a LinkedList (medium)

Given the head of a Singly LinkedList, write a method to modify the LinkedList such that the **nodes from the second half of the LinkedList are inserted alternately to the nodes from the first half in reverse order**. So if the LinkedList has nodes 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null, your method should return 1 -> 6 -> 2 -> 5 -> 3 -> 4 -> null.

Your algorithm should not use any extra space and the input LinkedList should be modified in-place.

Example 1:

```
Input: 2 -> 4 -> 6 -> 8 -> 10 -> 12 -> null
Output: 2 -> 12 -> 4 -> 10 -> 6 -> 8 -> null
```

Example 2:

```
Input: 2 -> 4 -> 6 -> 8 -> 10 -> null
Output: 2 -> 10 -> 4 -> 8 -> 6 -> null
```

Solution

This problem shares similarities with Palindrome LinkedList. To rearrange the given LinkedList we will follow the following steps:

1. We can use the **Fast & Slow pointers** method similar to Middle of the LinkedList to find the middle node of the LinkedList.
2. Once we have the middle of the LinkedList, we will reverse the second half of the LinkedList.
3. Finally, we'll iterate through the first half and the reversed second half to produce a LinkedList in the required order.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```python
from __future__ import print_function


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

  def print_list(self):
    temp = self
    while temp is not None:
      print(str(temp.value) + " ", end='')
      temp = temp.next
```

```
    print()


def reorder(head):
  if head is None or head.next is None:
    return

  # find middle of the LinkedList
  slow, fast = head, head
  while fast is not None and fast.next is not None:
    slow = slow.next
    fast = fast.next.next

  # slow is now pointing to the middle node
  head_second_half = reverse(slow)  # reverse the second half
  head_first_half = head

  # rearrange to produce the LinkedList in the required order
  while head_first_half is not None and head_second_half is not None:
    temp = head_first_half.next
    head_first_half.next = head_second_half
    head_first_half = temp

    temp = head_second_half.next
    head_second_half.next = head_first_half
    head_second_half = temp

  # set the next of the last node to 'None'
  if head_first_half is not None:
    head_first_half.next = None


def reverse(head):
  prev = None
  while head is not None:
    next = head.next
    head.next = prev
    prev = head
    head = next
  return prev


def main():
  head = Node(2)
  head.next = Node(4)
  head.next.next = Node(6)
  head.next.next.next = Node(8)
  head.next.next.next.next = Node(10)
  head.next.next.next.next.next = Node(12)
  reorder(head)
  head.print_list()


main()

```

Run Save Reset

Time Complexity

The above algorithm will have a time complexity of $O(N)$ where 'N' is the number of nodes in the LinkedList.

Space Complexity

The algorithm runs in constant space $O(1)$.

Back

Problem Challenge 2

Next

Problem Challenge 3

Completed

Report an Issue Ask a Question