Solution Review: Problem Challenge 1

We'll cover the following

Palindrome LinkedList (medium)

Given the head of a **Singly LinkedList**, write a method to check if the **LinkedList is a palindrome** or not.

Your algorithm should use **constant space** and the input LinkedList should be in the original form once the algorithm is finished. The algorithm should have $O(N)$ time complexity where 'N' is the number of nodes in the LinkedList.

Example 1:

```
Input: 2 -> 4 -> 6 -> 4 -> 2 -> null
Output: true
```

Example 2:

```
Input: 2 -> 4 -> 6 -> 4 -> 2 -> 2 -> null
Output: false
```

Solution

As we know, a palindrome LinkedList will have nodes values that read the same backward or forward. This means that if we divide the LinkedList into two halves, the node values of the first half in the forward direction should be similar to the node values of the second half in the backward direction. As we have been given a Singly LinkedList, we can't move in the backward direction. To handle this, we will perform 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.
3. Then, we will compare the first half with the reversed second half to see if the LinkedList represents a palindrome.
4. Finally, we will reverse the second half of the LinkedList again to revert and bring the LinkedList back to its original form.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

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


```

```python
def is_palindromic_linked_list(head):
  if head is None or head.next is None:
    return True

  # 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

  head_second_half = reverse(slow)  # reverse the second half
  # store the head of reversed part to revert back later
  copy_head_second_half = head_second_half

  # compare the first and the second half
  while (head is not None and head_second_half is not None):
    if head.value != head_second_half.value:
      break  # not a palindrome

    head = head.next
    head_second_half = head_second_half.next

  reverse(copy_head_second_half)  # revert the reverse of the second half

  if head is None or head_second_half is None:  # if both halves match
    return True

  return False


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(4)
  head.next.next.next.next = Node(2)

  print("Is palindrome: " + str(is_palindromic_linked_list(head)))

  head.next.next.next.next.next = Node(2)
  print("Is palindrome: " + str(is_palindromic_linked_list(head)))


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 1

Next

Problem Challenge 2

Completed

Report an Issue Ask a Question