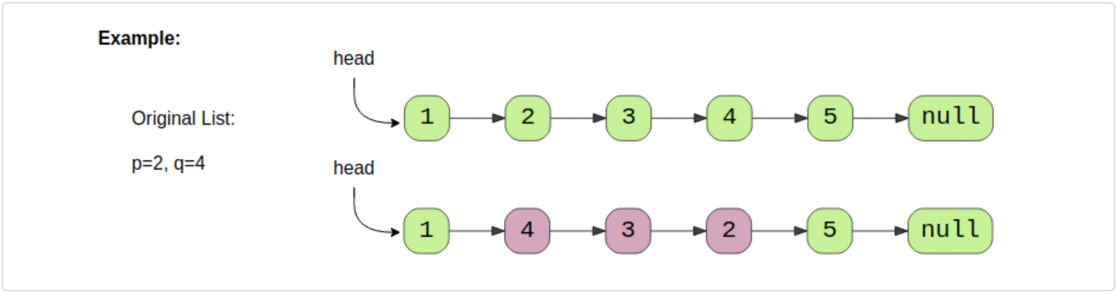
Reverse a Sub-list (medium)

We'll cover the following

Problem Statement

Given the head of a LinkedList and two positions ‘p’ and ‘q’, reverse the LinkedList from position ‘p’ to ‘q’.



Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```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(temp.value, end=" ")
            temp = temp.next
        print()


def reverse_sub_list(head, p, q):
    # TODO: Write your code here
    return head


def main():
    head = Node(1)
    head.next = Node(2)
    head.next.next = Node(3)
    head.next.next.next = Node(4)
    head.next.next.next.next = Node(5)

    print("Nodes of original LinkedList are: ", end='')
    head.print_list()
    result = reverse_sub_list(head, 2, 4)
```

```python
  print("Nodes of reversed LinkedList are: ", end='')
  result.print_list()


main()

```

Run Save Reset

Solution

The problem follows the **In-place Reversal of a LinkedList** pattern. We can use a similar approach as discussed in Reverse a LinkedList. Here are the steps we need to follow:

1. Skip the first `p-1` nodes, to reach the node at position `p`.
2. Remember the node at position `p-1` to be used later to connect with the reversed sub-list.
3. Next, reverse the nodes from `p` to `q` using the same approach discussed in Reverse a LinkedList.
4. Connect the `p-1` and `q+1` nodes to the reversed sub-list.

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(temp.value, end=" ")
      temp = temp.next
    print()


def reverse_sub_list(head, p, q):
  if p == q:
    return head

  # after skipping 'p-1' nodes, current will point to 'p'th node
  current, previous = head, None
  i = 0
  while current is not None and i < p - 1:
    previous = current
    current = current.next
    i += 1

  # we are interested in three parts of the LinkedList, the part before index 'p',
  # the part between 'p' and 'q', and the part after index 'q'
  last_node_of_first_part = previous
  # after reversing the LinkedList 'current' will become the last node of the sub-list
  last_node_of_sub_list = current
  next = None  # will be used to temporarily store the next node

  i = 0
  # reverse nodes between 'p' and 'q'
  while current is not None and i < q - p + 1:
    next = current.next
    current.next = previous
    previous = current
    current = next
    i += 1

  # connect with the first part
  if last_node_of_first_part is not None:
    # 'previous' is now the first node of the sub-list
    last_node_of_first_part.next = previous
  # this means p == 1 i.e., we are changing the first node (head) of the LinkedList
  else:
    head = previous

  # connect with the last part
  last_node_of_sub_list.next = current
  return head
```

```python


def main():
  head = Node(1)
  head.next = Node(2)
  head.next.next = Node(3)
  head.next.next.next = Node(4)
  head.next.next.next.next = Node(5)

  print("Nodes of original LinkedList are: ", end='')
  head.print_list()
  result = reverse_sub_list(head, 2, 4)
  print("Nodes of reversed LinkedList are: ", end='')
  result.print_list()


main()

```

Run Save Reset

Time complexity

The time complexity of our algorithm will be $O(N)$ where 'N' is the total number of nodes in the LinkedList.

Space complexity

We only used constant space, therefore, the space complexity of our algorithm is $O(1)$.

Similar Questions

Problem 1: Reverse the first 'k' elements of a given LinkedList.

Solution: This problem can be easily converted to our parent problem; to reverse the first 'k' nodes of the list, we need to pass `p=1` and `q=k`.

Problem 2: Given a LinkedList with 'n' nodes, reverse it based on its size in the following way:

1. If 'n' is even, reverse the list in a group of n/2 nodes.
2. If n is odd, keep the middle node as it is, reverse the first 'n/2' nodes and reverse the last 'n/2' nodes.

Solution: When 'n' is even we can perform the following steps:

1. Reverse first 'n/2' nodes: `head = reverse(head, 1, n/2)`
2. Reverse last 'n/2' nodes: `head = reverse(head, n/2 + 1, n)`

When 'n' is odd, our algorithm will look like:

1. `head = reverse(head, 1, n/2)`
2. `head = reverse(head, n/2 + 2, n)`

Please note the function call in the second step. We're skipping two elements as we will be skipping the middle element.

← Back
Reverse a LinkedList (easy)

Next →
Reverse every K-element Sub-list (me...

✓ Mark as Completed

Report an Issue   Ask a Question