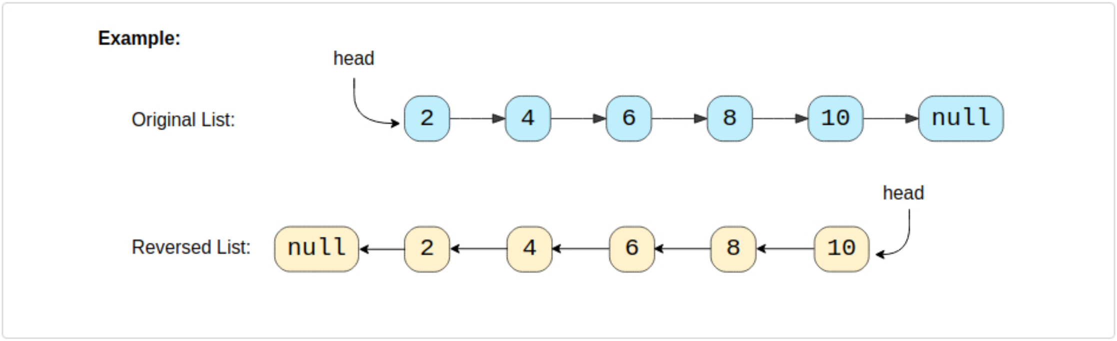
Reverse a LinkedList (easy)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
  - Code
  - Time complexity
  - Space complexity

Problem Statement

Given the head of a Singly LinkedList, reverse the LinkedList. Write a function to return the new head of the reversed LinkedList.



Try it yourself

Try solving this question here:

```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(head):
  # TODO: Write your code here
  return head


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)

  print("Nodes of original LinkedList are: ", end='')
```

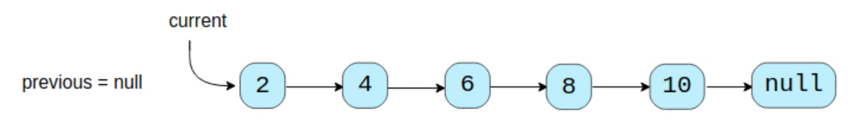
```
    head.print_list()
    result = reverse(head)
    print("Nodes of reversed LinkedList are: ", end='')
    result.print_list()


main()

```

Run Save Reset

Solution

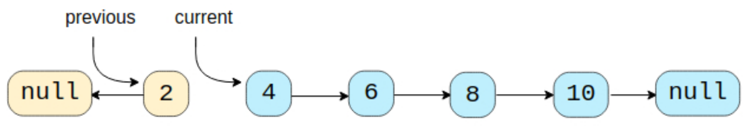
To reverse a LinkedList, we need to reverse one node at a time. We will start with a variable `current` which will initially point to the head of the LinkedList and a variable `previous` which will point to the previous node that we have processed; initially `previous` will point to `null`.

In a stepwise manner, we will reverse the `current` node by pointing it to the `previous` before moving on to the next node. Also, we will update the `previous` to always point to the previous node that we have processed. Here is the visual representation of our algorithm:

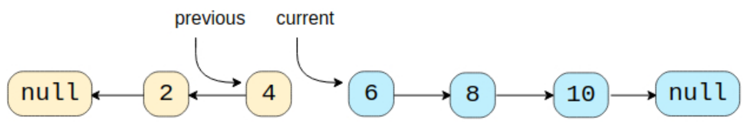


current = head, previous = null

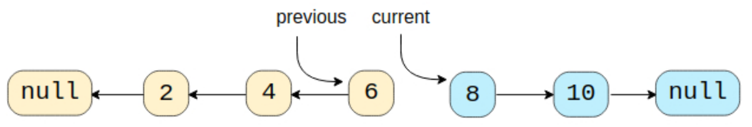
1 of 7



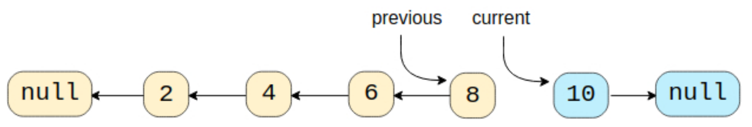
2 of 7



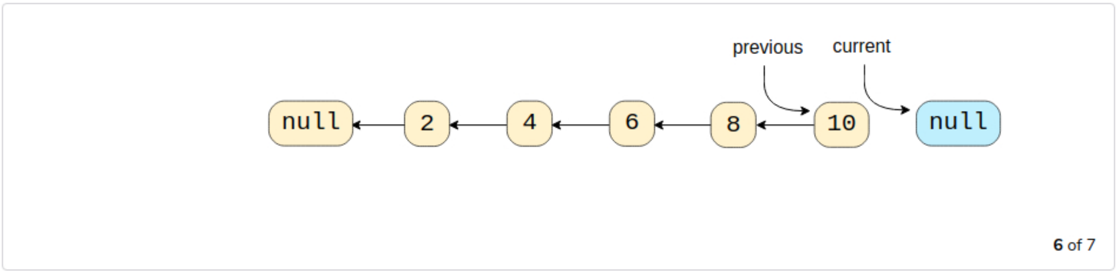
3 of 7



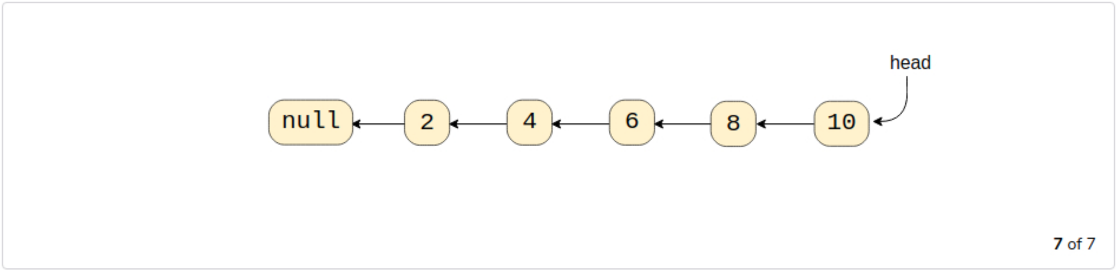
4 of 7



5 of 7



6 of 7



7 of 7

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(head):
  previous, current, next = None, head, None
  while current is not None:
    next = current.next  # temporarily store the next node
    current.next = previous  # reverse the current node
    previous = current  # before we move to the next node, point previous to the current node
    current = next  # move on the next node
  return previous


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)

  print("Nodes of original LinkedList are: ", end='')
  head.print_list()
  result = reverse(head)
  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

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 $O(1)$.

Back
Introduction

Next
Reverse a Sub-list (medium)

✓ Completed

Report an Issue Ask a Question