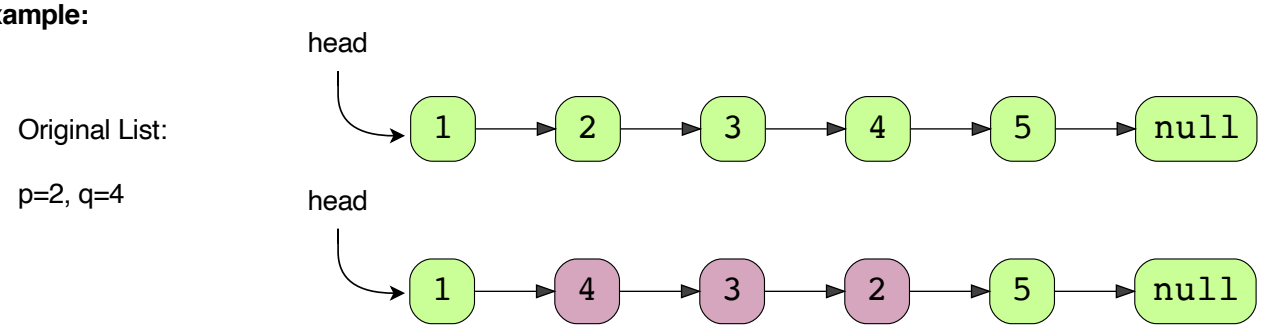
Reverse a Sub-list (medium)

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’.

Example:



Try it yourself#

Try solving this question here:

Java | Python3 | JS | C++

```
class Node {
  constructor(value, next=null){
    this.value = value;
    this.next = next;
  }

  get_list() {
    result = "";
    temp = this;
    while (temp !== null) {
      result += temp.value + " ";
      temp = temp.next;
    }
    return result;
  }
};

const reverse_sub_list = function(head, p, q) {
  // TODO: Write your code here
  return head;
};


head = new Node(1)
head.next = new Node(2)
head.next.next = new Node(3)
head.next.next.next = new Node(4)
head.next.next.next.next = new Node(5)

console.log(`Nodes of original LinkedList are: ${head.get_list()}`)
console.log(`Nodes of reversed LinkedList are: ${reverse_sub_list(head, 2, 4).g
et_list()}`)
```

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 (https://www.educative.io/collection/page/5668639101419520/56714648543

55968/4519653420302336/). 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 (https://www.educative.io/collection/page/5668639101419520/5671464854355968/4519653420302336/).
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

```
class Node {
  constructor(value, next = null) {
    this.value = value;
    this.next = next;
  }

  print_list() {
    let temp = this;
    while (temp !== null) {
      process.stdout.write(`${temp.value} `);
      temp = temp.next;
    }
    console.log();
  }
}


function reverse_sub_list(head, p, q) {
  if (p === q) {
```

```
    return head;
  }

  // after skipping 'p-1' nodes, current will point to 'p'th node
  let current = head,
    previous = null;
  let i = 0;
  while (current !== null && i < p - 1) {
    previous = current;
    current = current.next;
    i += 1;
  }

  // we are interested in three parts of the LinkedList, the part before index '
  // the part between 'p' and 'q', and the part after index 'q'
  const last_node_of_first_part = previous;
  // after reversing the LinkedList 'current' will become the last node of the su
  const last_node_of_sub_list = current;
  let next = null; // will be used to temporarily store the next node

  i = 0;
  // reverse nodes between 'p' and 'q'
  while (current !== null && 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 !== null) {
    // '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 Linl
  } else {
    head = previous;
  }

  // connect with the last part
  last_node_of_sub_list.next = current;
  return head;
}


const head = new Node(1);
head.next = new Node(2);
```

```
head.next.next = new Node(3);
head.next.next.next = new Node(4);
head.next.next.next.next = new Node(5);

process.stdout.write('Nodes of original LinkedList are: ');
head.print_list();
result = reverse_sub_list(head, 2, 4);
process.stdout.write('Nodes of reversed LinkedList are: ');
result.print_list();
```

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