811 23

View in Article (/articles/reverse-linked-list/)

206. Reverse Linked List

Notes

Description (/problems/reverse-linked-list/description/) Hints (/problems/reverse-linked-list/hints/) Submissions (/problems/rever

Quick Navigation

Solution

Approach #1 (Iterative) [Accepted]

Assume that we have linked list `1 → 2 → 3 → Ø`, we would like to change it to `Ø ← 1 ← 2 ← 3`.

While you are traversing the list, change the current node's next pointer to point to its previous element. Since a node does not have reference to its previous node, you must store its previous element beforehand. You also need another pointer to store the next node before changing the reference. Do not forget to return the new head reference at the end!

```
public ListNode reverseList(ListNode head) {
    ListNode prev = null;
    ListNode curr = head;
    while (curr != null) {
        ListNode nextTemp = curr.next;
        curr.next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}
```

Complexity analysis

- Time complexity : $O(n)$. Assume that n is the list's length, the time complexity is $O(n)$.
- Space complexity : $O(1)$.

Approach #2 (Recursive) [Accepted]

View in Article (/articles/reverse-linked-list/)

The recursive version is slightly trickier and the key is to work backwards. Assume that the rest of the list had already been reversed, now how do I reverse the front part? Let's assume the list is: $n_1 \rightarrow \ldots \rightarrow n_{k-1} \rightarrow n_k \rightarrow n_{k+1} \rightarrow \ldots \rightarrow n_m \rightarrow Ø$

Notes

Assume from node n_{k+1} to n_m had been reversed and you are at node n_k.

$n_1 \rightarrow \ldots \rightarrow n_{k-1} \rightarrow \mathbf{n_k} \rightarrow n_{k+1} \leftarrow \ldots \leftarrow n_m$

We want n_{k+1}'s next node to point to n_k.

So,

n_k.next.next = n_k;

Be very careful that n_1's next must point to Ø. If you forget about this, your linked list has a cycle in it. This bug could be caught if you test your code with a linked list of size 2.

```
public ListNode reverseList(ListNode head) {
    if (head == null || head.next == null) return head;
    ListNode p = reverseList(head.next);
    head.next.next = head;
    head.next = null;
    return p;
}
```

Complexity analysis

- Time complexity : $O(n)$. Assume that n is the list's length, the time complexity is $O(n)$.
- Space complexity : $O(n)$. The extra space comes from implicit stack space due to recursion. The recursion could go up to n levels deep.

Join the conversation

Signed in as **tan7**.

Post a Reply

jumaylisa commented 2 weeks ago

iscuss.leetcode.com/user/jumaylisa) View in Article (/articles/reverse-linked-list/)

@Liuerhu (https://discuss.leetcode.com/uid/434966) the second while loop, "p.next=null;" is need.

```
while(!stack.empty()){
p.next=stack.pop();
p=p.next;
p.next=null;
}
```

Liuerhu commented 3 weeks ago

iscuss.leetcode.com/user/liuerhu)

I want to know why my realization of Stack is timelimited for input[1,2]，seems not take that long

```
class Solution {
public ListNode reverseList(ListNode head) {
if(head==null) return null;
Stack<ListNode> stack=new Stack<>();
while(head!=null){
stack.push(head);
head=head.next;
}
ListNode nhead=stack.pop();
ListNode p=nhead;
while(!stack.empty()){
p.next=stack.pop();
p=p.next;
}
return nhead;
}
}
```

jessefeng commented last month

iscuss.leetcode.com/user/jessefeng)

View in Article (/articles/reverse-linked-list/)

Notes

```
reverseLinkedList = function (head) {
if(!head || !head.next ) {return head;}
let currentNode = head;
currentNode.prev = null;
while (currentNode.next) {
let oldNextNode = currentNode.next;
let newNextNode = currentNode.prev;
oldNextNode.prev = currentNode;
currentNode.next = newNextNode;
delete currentNode.prev;
currentNode = oldNextNode;
}
currentNode.next = currentNode.prev;
delete currentNode.prev;
return currentNode;
};
```

sikp commented last month

iscuss.leetcode.com/user/sikp)

@shwetas16 (https://discuss.leetcode.com/uid/436686) I have the same question.

geekysubham commented last month

iscuss.leetcode.com/user/geekysubham)

```
def reverse(self):
prev= None
curr=head
while curr:
prev,curr.next,curr= curr, prev,curr.next
return prev
```

shwetas16 commented 2 months ago

iscuss.leetcode.com/user/shwetas16)

What about pushing all values onto a stack and then building out the list as you pop from the stack? That would be O(n) time and O(n) space.

legbird3 commented 3 months ago

Got it. Never mind :)
iscuss.leetcode.com/user/legbird3)

View in Article (/articles/reverse-linked-list/)

legbird3 commented 3 months ago

Why space complexity for the first approach is O(1) not O(n)?
iscuss.leetcode.com/user/legbird3)

Notes

jlama commented 4 months ago

iscuss.leetcode.com/user/jlama)

```
public ListNode reverseList(ListNode head) {
    ListNode curr = head;
    ListNode prev = null;
    ListNode nextN = null;
    while(curr != null) {
        nextN = curr.next;
        curr.next = prev;
        prev = curr;
        curr = nextN;
    }
    head = prev;
    return head;
}
```

RF commented 4 months ago

iscuss.leetcode.com/user/rf) View in Article (/articles/reverse-linked-list/)

```
class Solution(object):
    def reverseListIterative(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        rev, cur = None, head
        while cur:
            rev, rev.next, cur = cur, rev, cur.next
        return rev
    # The recursive version is slightly trickier and the key is to work backwards.
    def reverseListRecursive(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head or not head.next:
            return head
        rev = self.reverseListRecursive(head.next)
        head.next.next = head # Reverse the tail of new reversed to list to point to cu
        head.next = None
        return rev
```

Notes

View original thread (https://discuss.leetcode.com/topic/28)

Load more comments...