83. Remove Duplicates from Sorted List

April 18, 2016 linked-list (/articles/?tag=linked-list)

Question | Editorial Solution

Question

> Given a sorted linked list, delete all duplicates such that each element appear only *once*.
>
> For example,
> Given `1->1->2`, return `1->2`.
> Given `1->1->2->3->3`, return `1->2->3`.

Solution

Straight-Forward Approach [Accepted]

Algorithm

This is a simple problem that merely tests your ability to manipulate list node pointers. Because the input list is sorted, we can determine if a node is a duplicate by comparing its value to the node *after* it in the list. If it is a duplicate, we change the `next` pointer of the current node so that it skips the next node and points directly to the one after the next node.

Java

```
public ListNode deleteDuplicates(ListNode head) {
    ListNode current = head;
    while (current != null && current.next != null) {
        if (current.next.val == current.val) {
            current.next = current.next.next;
        } else {
            current = current.next;
        }
    }
    return head;
}
```

Complexity Analysis

Because each node in the list is checked exactly once to determine if it is a duplicate or not, the total run time is $O(n)$, where n is the number of nodes in the list.

Space complexity is $O(1)$ since no additional space is used.

Correctness

We can prove the correctness of this code by defining a *loop invariant*. A loop invariant is condition that is true before and after every iteration of the loop. In this case, a loop invariant that helps us prove correctness is this:

> All nodes in the list up to the pointer `current` do not contain duplicate elements.

We can prove that this condition is indeed a loop invariant by induction. Before going into the loop, `current` points to the head of the list. Therefore, the part of the list up to `current` contains only the head. And so it can not contain any duplicate elements. Now suppose `current` is now pointing to some node in the list (but not the last element), and the part of the list up to `current` contains no duplicate elements. After another loop iteration, one of two things happen.

1. `current.next` was a duplicate of `current`. In this case, the duplicate node at `current.next` is deleted, and `current` stays pointing to the same node as before. Therefore, the condition still holds; there are still no duplicates up to `current`.

2. `current.next` was not a duplicate of `current` (and, because the list is sorted, `current.next` is also not a duplicate of any other element appearing *before* `current`). In this case, `current` moves forward one step to point to `current.next`. Therefore, the condition still holds; there are no duplicates up to `current`.

At the last iteration of the loop, `current` must point to the last element, because afterwards, `current.next = null`. Therefore, after the loop ends, all elements up to the last element do not contain duplicates.

Analysis written by: @noran.

Previous (/articles/symmetric-tree/) Next (/articles/number-1-bits/)

lttzzlll commented 2 months ago

this is the simplest approch i have seen.

iscuss.leetcode.com/user/lttzzlll)

kkamaraj commented 7 months ago

@EnzoHarris (https://discuss.leetcode.com/uid/73496) you are right! Thanks for the clarification.

iscuss.leetcode.com/user/kkamaraj)

EnzoHarris commented 7 months ago

@kkamaraj (https://discuss.leetcode.com/uid/73279) it's sorted list.

iscuss.leetcode.com/user/enzoharris)

kkamaraj commented 8 months ago

This solution would solve if the duplicates are next to each other. What if I have linked list that looks like 1->2->3->1->4?

iscuss.leetcode.com/user/kkamaraj)

jasonvfang commented 8 months ago

iscuss.leetcode.com/user/jasonvfang)

```
struct ListNode* deleteDuplicates(struct ListNode* head) {
    if(head == NULL)
        return NULL;

    struct ListNode* p = NULL, *q = NULL;

    p = head;
    q = p->next;

    while(q)
    {
        struct ListNode *tmp = q->next;

        if (!(p->val ^ q->val))
        {
            //same element
            p->next = tmp;
            free(q);
            q = tmp;
        }
        else
        {
            p = q;
            q = q->next;
        }
    }

    return head;
}
```

babyLiyuan commented 9 months ago

There are two methods to solve this problem, recursive one and non-recursive one. The method in your article is recursive, which is O(1) space and O(n) time. The non-recursive one which is not given is O(n) space and O(n) time.

iscuss.leetcode.com/user/babyliyuan)

nrl commented 11 months ago

@dirty_ninja (https://discuss.leetcode.com/uid/47533), Manually 'free'ing up memory is only applicable in C/C++. It is not required in Java or Python.

iscuss.leetcode.com/user/nrl)

dirty_ninja commented last year

@1337c0d3r (https://discuss.leetcode.com/uid/1), Do you mean we have to delete the space of the "duplicate" elements?

iscuss.leetcode.com/user/dirty_ninja)

xuzhenqi1993 commented last year

@1337c0d3r (https://discuss.leetcode.com/uid/1) That depends on the way the memory allocated.

iscuss.leetcode.com/user/xuzhenqi1993)

coolseraz commented last year

You can use Java. Put all elements in a LinkedHashSet (maintains order even in a set). Then put all of these elements from the Set into a different linked list and return it.

iscuss.leetcode.com/user/coolseraz)

View original thread (https://discuss.leetcode.com/topic/68)

Load more comments...