Problems On Linked List

Created	@August 19, 2022 8:32 PM
Class	
Type	
Materials	
Reviewed	☐

Problem 1:

Remove Duplicates from Sorted List - LeetCode

Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well. Example 1: Input: head = [1,1,2]Output: [1,2] Example 2: Input:

https://leetcode.com/problems/remove-duplicates-from-sorted-list/

Solution:

Brute force:

We store all the unique elements in a set by traversing the original list, and then create a new linked list using only unique values and return the head of it.

In this approach the space complexity: O(n) and we are also returning a new Linked List.

How can we optimise ?

Now we need to focus on the given input, the given linked list is sorted. If it would have an unsorted list then it would have become a bit difficult for us to actually remove the duplicates without storing them, but because the data is sorted and we have a linked list we can do something better.

In a linked list if we want to remove a node somewhere in between the list and if we have access to the prev node, then we can remove it in O(1). This would not have been possible in a normal array in O(1) time.

Now we know that data is sorted, so any repetition of any node will be present just next to it. So what we can do we can start iterating on the Linked List, and when we are sitting on a current node, we can check the value of the next node, if both of them have got the same value, we can delete the next node in O(1) time. The moment we find that data of the next node is now no more equal to the current node, we can say that we have removed all the duplicates of the current node. And now we can do the same for next possible nodes.

```
var deleteDuplicates = function(head) {
// Time: O(n)
    let temp = head;
    while(temp != null && temp.next != null) {
        // loop till we dont reach the tail or surpass the tail as there wont be duplicate
s after tail
        let nextNode = temp.next;
        while(nextNode != null && temp.val == nextNode.val) {
            // this loop removes the duplicate
            temp.next = nextNode.next;
            nextNode.next = null;
            nextNode = temp.next;
        }
        temp = temp.next;
    }
    return head;
};
```

Problem 2:

Merge Two Sorted Lists - LeetCode

You are given the heads of two sorted linked lists list1 and list2. Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of

https://leetcode.com/problems/merge-two-sorted-lists/

LeetCode

Solution:

Brute Force:

We can simply iterate over both the linked lists, store their data in an array one after another, then sort the array and then using the sorted array's data we can create a new ll.

Time: O((n+m)log(n+m))

Space: O(n+m)

How can we optimise ?

In order to solve this problem in a more optimised way, we can use the strategy of merge two sorted arrays that is sub-routine of merge sort. In merge two sorted array, we take two pointers put them on each array and then compare the values and based on the comparison insert the values in a new sorted array.

So what we can do here is, we can have to heads as the two pointers and a new result LL, then we can compare the values of the heads and based on that one by one add values to the new LL by create new nodes.

Time: O(n+m)

Space: O(n+m)

Can we optimise over space ?

Because of the fact that it is a LL, instead of inserting new nodes in a new LL, we can manipulate the nodes of the old LLs, as well so that whenever we compare two nodes and one node is less than the other, then we can remove the node from it's original list and attach to result list.

Problem 3:

Given a linked List, in a single pass, try to find it's middle node.

1 → 2 → 3 → 4 → 5 → 6 → x

Solution:

Assume that we have two players, P1 and P2, where P1 has a speed of x whereas P2 has speed of 2x.

Assume that they both run on a track for t units of time, and we know that P2 covered n units of distance, then how much distance P1 covered ?

P1 will cover n/2 distance

Let’s try to apply similar logic here, i.e. let’s say we have a fast node and a slow node, in every iteration, make fast node move two nodes ahead, and slow node move one node ahead. Then when the fast node reaches the tail or surpasses the tail, slow node is at the mid.

```
var middleNode = function(head) {
    let slow = head;
    let fast = head;
    while(fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
};
```

Problem 4:

Swapping Nodes in a Linked List - LeetCode

You are given the head of a linked list, and an integer k. Return the head of the linked list after swapping the values of the node from the beginning and the node from the end (the list is 1-indexed).

https://leetcode.com/problems/swapping-nodes-in-a-linked-list/

LeetCode

Solution:

We can have a very interesting observation here, if we have access to the kth node from left (which is easy too right !) then the distance between head and this node is K.

So if we mark two nodes A and B such that A is at the head and B is at the Kth node from left, and then every time move both A and B to their next node, then when B reaches the tail, A is at the kth node from right.

Apart from the kth node from left and right, we need to maintain their prev node as well, then only we can manipulate the pointers correctly.