Palindrome Linked List

Given a singly linked list, determine if it is a palindrome.

Follow up:
Could you do it in O(n) time and O(1) space?

Solution 1

It is a common misunderstanding that the space complexity of a program is just how much the size of additional memory space being used besides input. An important prerequisite is neglected the above definition: the input has to be read-only. By definition, changing the input and change it back is not allowed (or the input size should be counted when doing so). Another way of determining the space complexity of a program is to simply look at how much space it has written to. Reversing a singly linked list requires writing to O(n) memory space, thus the space complexities for all "reverse-the-list"-based approaches are O(n), not O(1).

Solving this problem in O(1) space is theoretically impossible due to two simple facts: (1) a program using O(1) space is computationally equivalent to a finite automata, or a regular expression checker; (2) the pumping lemma states that the set of palindrome strings does not form a regular set.

Please change the incorrect problem statement.

written by wangmenghui original link here

Solution 2

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        if(head==NULL||head->next==NULL)
            return true;
        ListNode* slow=head;
        ListNode* fast=head;
        while(fast->next!=NULL&&fast->next->next!=NULL){
            slow=slow->next;
            fast=fast->next->next;
        }
        slow->next=reverseList(slow->next);
        slow=slow->next;
        while(slow!=NULL){
            if(head->val!=slow->val)
                return false;
            head=head->next;
            slow=slow->next;
        }
        return true;
    }
    ListNode* reverseList(ListNode* head) {
        ListNode* pre=NULL;
        ListNode* next=NULL;
        while(head!=NULL){
            next=head->next;
            head->next=pre;
            pre=head;
            head=next;
        }
        return pre;
    }
};
```

written by YCG09 original link here

Solution 3

O(n) time, O(1) space. The second solution restores the list after changing it.

Solution 1: *Reversed first half == Second half?*

Phase 1: Reverse the first half while finding the middle.
Phase 2: Compare the reversed first half with the second half.

```
def isPalindrome(self, head):
    rev = None
    slow = fast = head
    while fast and fast.next:
        fast = fast.next.next
        rev, rev.next, slow = slow, rev, slow.next
    if fast:
        slow = slow.next
    while rev and rev.val == slow.val:
        slow = slow.next
        rev = rev.next
    return not rev
```

Solution 2: *Play Nice*

Same as the above, but while comparing the two halves, restore the list to its original state by reversing the first half back. Not that the OJ or anyone else cares.

```
def isPalindrome(self, head):
    rev = None
    fast = head
    while fast and fast.next:
        fast = fast.next.next
        rev, rev.next, head = head, rev, head.next
    tail = head.next if fast else head
    isPali = True
    while rev:
        isPali = isPali and rev.val == tail.val
        head, head.next, rev = rev, head, rev.next
        tail = tail.next
    return isPali
```

written by StefanPochmann original link here