Reverse Linked List

Reverse a singly linked list.

click to show more hints.

Hint:
A linked list can be reversed either iteratively or recursively. Could you implement both?

Solution 1

```
// iterative solution

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

// recursive solution

```
public ListNode reverseList(ListNode head) {
    return reverseListInt(head, null);
}

public ListNode reverseListInt(ListNode head, ListNode newHead) {
    if(head == null)
        return newHead;
    ListNode next = head.next;
    head.next = newHead;
    return reverseListInt(next, head);
}
```

written by braydenCN original link here

Solution 2

xWell, since the `head` pointer may also be modified, we create a `new_head` that points to it to facilitate the reverse process.

For the example list `1 -> 2 -> 3 -> 4 -> 5` in the problem statement, it will become `0 -> 1 -> 2 -> 3 -> 4 -> 5` (we init `new_head -> val` to be `0`). Then we set a pointer `pre` to `new_head` and another `cur` to `head`. Then we keep inserting `cur -> next` after `pre` until `cur` becomes the last node. The code is follows.

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* new_head = new ListNode(0);
        new_head -> next = head;
        ListNode* pre = new_head;
        ListNode* cur = head;
        while (cur && cur -> next) {
            ListNode* temp = pre -> next;
            pre -> next = cur -> next;
            cur -> next = cur -> next -> next;
            pre -> next -> next = temp;
        }
        return new_head -> next;
    }
};
```

This link provides a more concise solution without using the `new_head`. The idea is to reverse one node at a time for the beginning of the list. The rewritten code is as follows.

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* pre = NULL;
        while (head) {
            ListNode* next = head -> next;
            head -> next = pre;
            pre = head;
            head = next;
        }
        return pre;
    }
};
```

Well, both of the above solutions are iterative. The hint has also suggested us to use recursion. In fact, the above link has a nice recursive solution, whose rewritten code is as follows.

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if (!head || !(head -> next)) return head;
        ListNode* node = reverseList(head -> next);
        head -> next -> next = head;
        head -> next = NULL;
        return node;
    }
};
```

The basic idea of this recursive solution is to reverse all the following nodes after `head`. Then we need to set `head` to be the final node in the reversed list. We simply set its next node in the original list (`head -> next`) to point to it and sets its `next` to be `NULL`.

written by jianchao.li.fighter original link here

Solution 3

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

    struct ListNode *p=head;
    p=head->next;
    head->next=NULL;
    while(NULL!=p){
        struct ListNode *ptmp=p->next;
        p->next=head;
        head=p;
        p=ptmp;
    }
    return head;
}
```

above is the iterative one. simple, nothing to explain.

```
struct ListNode* reverseListRe(struct ListNode* head) {
    if(NULL==head||NULL==head->next) return head;

    struct ListNode *p=head->next;
    head->next=NULL;
    struct ListNode *newhead=reverseListRe(p);
    p->next=head;

    return newhead;
}
```

above is the recursively one.Both are accepted.

written by redace85 original link here