Author: Xiaoqiang Yan
Date: 08/06/2018

LeetCode 143. Reorder List

Given a singly linked list *L*: *L*0→*L*1→…→*Ln*-1→*L*n,

reorder it to: *L*0→*Ln*→*L*1→*Ln*-1→*L*2→*Ln*-2→…

You may not modify the values in the list's nodes, only nodes itself may be changed.

Example 1:

```
Given 1->2->3->4, reorder it to 1->4->2->3.
```

Example 2:

```
Given 1->2->3->4->5, reorder it to 1->5->2->4->3.
```

Method

1. Brute Force

we can save all numbers in the list in an array, traverse the array from tail, and insert nth node behind 1st node. repeat this operation until we meet the middle node.
Although there's no space limit, but the 1st solution is not accepted.
2. T(n) = O(n), space is O(1)
Similar with leetcode 234, we can find the middle node first, reverse the latter half list, and the reorder the original list.

Code

```
class ListNode {
        int val;
        ListNode *next;
        ListNode(int x): val(x), next(NULL) {}
};

void reorderList( ListNode *head) {
        if(head || head->next) return ;
        vector<int> num;
        ListNode* p = head;
        while(p){
                num.push_back(p->val);
        }
        p = head;
        int k = num.size()-1;
        while(p) {
                if(k==num.size()/2) break;
                ListNode *cur =  new ListNode(num[k]);
                ListNode *temp = p->next;
                cur = p->next;
                cur->next = temp;
                p = temp;
                k--;
                delete cur;
        }
        num.size()%2? p->next=NULL: p->next->next =NULL;
        //delete cur;
}

void reorderList( ListNode* head) {
        if(head==NULL || head->next == NULL) return;
        //find the middle node
        ListNode* mid = head, tail = head;
        while(tail && tail->next) {
                tail = tail->next->next;
                mid = mid->next;
        }
        //reverse the latter half
        ListNode* pre = NULL, *cur = mid;
        while(cur) {
                ListNode* temp = cur->next;
                cur->next = pre;
                pre = cur;
                cur = temp;
        }
        //reorder the list
        ListNode *pf= head, *pl = pre;
        while(pf ->next!= NULL && pl ->next!= NULL) { // here the condition should be changed,
                ListNode* temp1 = pf->next;
                ListNode* temp2 = pl->next;
                pf ->next = pl;
                pl->next = temp1;
                pf = temp1;
                pl = temp2;
                temp1 = temp1->next;
                temp2 = temp2->next;
        }
}
```

Note

after reversing the latter half list, if we need to compare the two half lists, the condition is (pf !=NULL && pl != NULL), but if we have temp pointer to next, we should consider if pf != NULL but pf->next == NULL, if we keep going, the next doesn't have next because of NULL. So we should change the condition to be (pf ->next != NULL && pl->next != NULL)

Summary

1. Double pointer model
2. for linked list, reverse latter half list is a good way to save space.