高效面试之leetcode链表题汇总

2014-10-26 01:21 691人阅读 评论(0) 收藏[1] 举报

高效面试宝典（*15*）

题目索引：

1.偶数位逆序插入reoder list

Given `{1,2,3,4}`, reorder it to `{1,4,2,3}`

2.交换相邻的二个节点 swap nodes in pairs

Given `1->2->3->4`, you should return the list as `2->1->4->3`

3.删除指定位置的节点

删除重复节点

5.给定数组，小于该值的放左边，大于该值的放右边

6.链表递归反转

7.判断链表是否有环

8.链表归并排序

一、解题思路

1.判断head是否为空或者有时判断是否为1个

if(head==NULL || head->next=NULL) return head;

2.是否需要头结点dummy

ListNode dummy(-1);

dummy.next=head;

left_cur=&dummy;

prev =&dummy;

3.遍历遍历时，需要几个指针

<1>一个型 ptr

<2>两个型 left_cur和right_cur fast和slow

exp1.一个链表按值分成2个：

ListNode left_dummy(-1);//头结点

ListNode right_dummy(-1);

ListNode *left_cur=&left_dummy;//这一句相当的重要

ListNode *right_cur=&right_dummy;

ptr=head;

while(ptr)

if(ptr->val<x){

left_cur->next=ptr;

left_cur=ptr;//别忘了

else{

right_cur->next=ptr;

right_cur=ptr;

ptr=ptr->next;

exp2.slow和fast把链表按中点划分为2个

```
while(fast && fast->next)
    prev=slow;
    slow=slow->next;
    fast=fast->next->next;
prev->next=NULL;
```

<3>三个型 prev, pcur,pnext

典型应该，链表逆序

```
 ListNode* reverse(ListNode *head) {
     if(head == NULL || head->next == NULL ) return head;
    ListNode *prev=NULL,*pcur=NULL,*pnext=NULL;
    pcur=head;//循环开始前，pcur指向第一个节点
    while(pcur)
  pnext=pcur->next;
        pcur->next=prev;
prev=pcur;//有先后顺序，第一次时把pcur=pnext写在这里了，导致出错
        pcur=pnext;
    return prev;//循环完后,prev是指向的最后一个节点
```

4.fast指针

```
fast=head;
while (fast && fast->next
    fast=fast->next->next;
```

1->2

一次循环后,fast为空

1->2->3

一次循环后:fast指向3 fast->next为空

循环体都只循环执行一次，也就是说如果链表长度为奇数，要求循环执行次数为(n-1)/2，这样刚好满足

二.链表题目汇总

Given a singly linked list : 0→1→…→-1→n,
reorder it to: 0→→1→-1→2→-2→…

You must do this in-place without altering the nodes' values.

For example,
Given `{1,2,3,4}`, reorder it to `{1,4,2,3}`.

1.通过slow和fast找到链表从中点,fast到底时，slow指针的位置即为中点

2.把第二段逆序，即slow指针指向的后一段

3.合并二段

```
 * Definition for singly-linked list.

 * struct ListNode {

 * int val;

 * ListNode *next;

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

class Solution {

public:

void reorderList(ListNode *head) {
```

```
    if (head == nullptr || head->next == nullptr) return;
    ListNode *slow = head, *fast = head, *prev = nullptr;
while (fast && fast->next)
        prev = slow;
        slow = slow->next;
        fast = fast->next->next;
    prev->next = nullptr; // 分成2端，slow为前面一部分
    slow = reverse(slow);
    // merge two lists
    ListNode *curr = head;
    while (curr->next)
        ListNode *tmp = curr->next;
        curr->next = slow;
        slow = slow->next;
        curr->next->next = tmp;
        curr = tmp;
    curr->next = slow;
 ListNode* reverse(ListNode *head) {
     if(head == NULL || head->next == NULL ) return head;
    ListNode *prev=NULL,*pcur=NULL,*pnext=NULL;
    pcur=head;
    while(pcur)
```

```
 pnext=pcur->next;
      pcur->next=prev;
prev=pcur;//有先后顺序，第一次时把pcur=pnext写在这里了，导致出错
      pcur=pnext;
   return prev;//循环完后,prev是指向的最后一个节点
      if(head == NULL || head->next == NULL ) return head;
      ListNode *prev=NULL,*pcur=NULL,*pnext=NULL;
      pcur=head;
      while(pcur)
         pnext=pcur->next;
         pcur->next=prev;
         pcur=pnext;
         prev=pcur;
      return prev;//循环完后,prev是指向的最后一个节点
```

题目：Given a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

Follow up:
Can you solve it without using extra space?

1.通过fast,slow指针找到环中的交点

2.从交点处断开，变为2条链表，链表1从交点p开始，链表2从head开始

3.链表1长度为len1，链表2长度为len2，如果len2>len1,则链表2的头从head+(len2-len1)开始。第一个交点，则为环的入口。

当 fast 与 slow 相遇时，slow 肯定没有遍历完链表，而 fast 已经在环内循环了 n 圈 (1 n)。假

设 slow 走了 s 步，则 fast 走了 2s 步（fast 步数还等于 s 加上在环上多转的 n 圈），设环长为 r，则：

2s = s + nr

设整个链表长 L，环入口点与相遇点距离为 a，起点到环入口点的距离为 x，则

x + a = nr = (n−1)r + r = (n -)r + L- x

x = (n-1)r + (L−x−a)

L−x−a 为相遇点到环入口点的距离，由此可知，从链表头到环入口点等于 n 1 圈内环 + 相遇

点到环入口点，于是我们可以从 head 开始另设一个指针 slow2，两个慢指针每次前进一步，它俩

一定会在环入口点相遇

也就是说如果选择相交点作为slow1, head作为slow2. 虽然slow1可能会在环上走多圈，但是那么相交点一定在环入口。

```
 * Definition for singly-linked list.

 * struct ListNode {

 * int val;

 * ListNode *next;

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

class Solution {

public:

    ListNode *detectCycle(ListNode *head) {

        if(head == NULL ) return NULL;

        ListNode *slow=head,*fast=head;

        while(fast && fast->next)
```

```
        fast=fast->next->next;
        slow=slow->next;
        if(slow == fast)
            ListNode *L1=slow;
            ListNode *L2=head;
            while(L1 != L2)
                L1=L1->next;
                L2=L2->next;
            return L2;
    return NULL;
```

题目：Given a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

Follow up:
Can you solve it without using extra space?

1.通过fast,slow指针找到环中的交点

2.从交点处断开，变为2条链表，链表1从交点p开始，链表2从head开始

3.链表1长度为len1，链表2长度为len2，如果len2>len1,则链表2的头从head+(len2-len1)开始。第一个交点，则为环的入口。

当 fast 与 slow 相遇时，slow 肯定没有遍历完链表，而 fast 已经在环内循环了 n 圈 (1 n)。假

设 slow 走了 s 步，则 fast 走了 2s 步（fast 步数还等于 s 加上在环上多转的 n 圈），设环长为 r，则：

2s = s + nr

设整个链表长 L，环入口点与相遇点距离为 a，起点到环入口点的距离为 x，则

x + a = nr = (n−1)r + r = (n -)r + L- x

x = (n-1)r + (L−x−a)

L−x−a 为相遇点到环入口点的距离，由此可知，从链表头到环入口点等于 n 1 圈内环 + 相遇

点到环入口点，于是我们可以从 head 开始另设一个指针 slow2，两个慢指针每次前进一步，它俩

一定会在环入口点相遇

也就是说如果选择相交点作为slow1, head作为slow2. 虽然slow1可能会在环上走多圈，但是那么相交点一定在环入口。

```
 * Definition for singly-linked list.

 * struct ListNode {

 * int val;

 * ListNode *next;

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

class Solution {

public:

    ListNode *detectCycle(ListNode *head) {

        if(head == NULL ) return NULL;

        ListNode *slow=head,*fast=head;

        while(fast && fast->next)

            fast=fast->next->next;

            slow=slow->next;

            if(slow == fast)

                ListNode *L1=slow;

                ListNode *L2=head;
```

```
        while(L1 != L2)
            L1=L1->next;
            L2=L2->next;
        return L2;
    return NULL;
```

Given a linked list, swap every two adjacent nodes and return its head.

For example,
Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may modify the values in the list, only nodes itself can be changed.

1.把链表分成2部分,left_dummy->1->3->5 right_dummy->2->4 注意是需要头结点的

2.只把两部分给连接起来，如果是奇数个的话，也没关系，反正left链表要么和right长度相等，要么多一，所以while只需要判断right链表

3.注意特色情况，链表为0或者为1

```
class Solution {
public:
    ListNode *swapPairs(ListNode *head) {
        if(head == NULL) return NULL;
        if(head->next==NULL) return head;
        ListNode left_dummy(-1);
        ListNode right_dummy(-1);
ListNode *left_cur=&left_dummy,*right_cur=&right_dummy;
        ListNode *ptr=head;
```

```
    int i=1;
    while(ptr)
      if(i%2==1)
        left_cur->next=ptr;
        left_cur=ptr;
        right_cur->next=ptr;
        right_cur=ptr;
      ptr=ptr->next;
    left_cur=left_dummy.next;
    right_cur=right_dummy.next;
    while(right_cur)
        ListNode *tmp1=right_cur->next;
        ListNode *tmp2=left_cur->next;
        right_cur->next=left_cur;
        left_cur->next=tmp1;
        right_cur=tmp1;
        left_cur=tmp2;
    return right_dummy.next;
```

Given a linked list, remove the nth node from the end of list and return its head.

For example,

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.

Note:

Given n will always be valid.

Try to do this in one pass.

1,要求在一个过程中完成，可以用2个指针fast,slow. fast指针先走n步，到fast走到尾巴时，slow走到倒数第n个

2.第一次，错了，{1} 1 这种测试用例通不过，所以需要一个头结点dummy, head->1,保证链表至少有2个节点，why

```
class Solution {

public:

    ListNode *removeNthFromEnd(ListNode *head, int n) {

        if(head==NULL) return NULL;

        ListNode dummy(-1);//针对1个元素的测试用例，相当重要

        dummy.next=head;

        ListNode *fast=&dummy,*slow=&dummy;

        for(int i=0;i<n;i++)

            fast=fast->next;

        while(fast->next)

            slow=slow->next;

            fast=fast->next;

        ListNode *tmp=slow->next;

        slow->next=slow->next->next;

        delete tmp;

        return dummy.next;
```

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers

from the original list.

For example,

Given 1->2->3->3->4->4->5, return 1->2->5.

Given 1->1->1->2->3, return 2->3

在I 答案中，直接用了 delete temp来删除节点，我没有用，是通过next指针越过要删除的节点来操作，没有free/delete掉。这一题可以通过增加一个变量，来说明当前节点是否重复。因为当前节点pcur要被删除，所以也多需要一个prev指针。

```
class Solution {

public:

    ListNode *deleteDuplicates(ListNode *head) {

        if( head == NULL ) return NULL;

        ListNode dummy(-1);//头结点,像第二例子那样，就需要头节点

        dummy.next=head;

        ListNode *prev=&dummy,*pcur=head;

        while(pcur)

            int flag=0;//为1则当前节点重复

            while(pcur->next && pcur->val == pcur->next->val)

                flag=1;

                pcur=pcur->next;//重复节点的最后一个

            if(flag == 1)

                prev->next=pcur->next;
```

```
        prev=pcur;

        pcur=pcur->next;

    return dummy.next;
```

Given a linked list and a value , partition it such that all nodes less than come before nodes greater than or equal to .

You should preserve the original relative order of the nodes in each of the two partitions.

For example,
Given `1->4->3->2->5->2` and = 3,
return `1->2->2->4->3->5`.

分析：快速排序就是这样，首先选择一个基本点，一趟比较下来之后，基准点左边全是小于其的数，基本点右边全是大于其本身的数。然后递归调用确定左右边界qsort(a[],left,right)，就完成了排序。

本题是链表，链表不是直接下表访问，只能next往前，我们也可以用类似于qsort的partition的思想，用两根指针记录左右两边。

循环执行后，在合并左右指针。

left指针 1->2->2

right指针 4->3->5

一个技巧：因为while循环是left_cur最后是指向链表的最后一个节点，当操作完毕后，我们需要找到链表头部来使用整个链表。这样的话，我们可以：使用带头结点的链表

1）声明头结点，不是指针，以后取名的话用dummy吧

ListNode left_dummy(-1);//头结点

2)获取dummy节点的地址，赋值给循环移动的指针

 ListNode *left_cur=&left_dummy;

3)获取链表

left_dummy.next

head节点->1->2->2

head节点->4->3->5

```
class Solution {
public:
    ListNode *partition(ListNode *head, int x) {
        if(head==NULL) return NULL;
        ListNode *ptr=NULL;
        ListNode left_dummy(-1);//头结点
        ListNode right_dummy(-1);
        ListNode *left_cur=&left_dummy;//这一句相当的重要,自己当时不会写,其实就是做了2个没有值的头结点,这样left_dummy.next就是左链表
        ListNode *right_cur=&right_dummy;
         ptr=head;
        while(ptr)
           if(ptr->val<x){
               left_cur->next=ptr;
               left_cur=ptr;
           else{
               right_cur->next=ptr;
               right_cur=ptr;
           ptr=ptr->next;
        left_cur->next=right_dummy.next;
```

right_cur->next=NULL;

return left_dummy.next;

链表递归反转

```
1. //递归方式
2. ListNode * ReverseList(ListNode * head)
3.    //如果链表为空或者链表中只有一个元素
4.    if(head==NULL || head->next==NULL)
5.        return head;
6.       ListNode * newhead=ReverseList2(head->next);//先反转后面的链表
7.       head->next->next=head;//再将当前节点设置为其然来后面节点的后续节点
8.       head->next=NULL;  //链表末尾置为NULL
9.       return newhead;
```

链表归并排序

```
public:
    ListNode *sortList(ListNode *head) {
        if (head == NULL || head->next == NULL)return head;

        // 快慢指针找到中间节点
        ListNode *fast = head, *slow = head;
        while (fast->next != NULL && fast->next->next != NULL) {
            fast = fast->next->next;
            slow = slow->next;
        }
        // 断开
        fast = slow;
        slow = slow->next;
        fast->next = NULL;

        ListNode *l1 = sortList(head);  // 前半段排序
        ListNode *l2 = sortList(slow);  // 后半段排序
        return mergeTwoLists(l1, l2);
    }

    // Merge Two Sorted Lists
    ListNode *mergeTwoLists(ListNode *l1, ListNode *l2) {
        ListNode dummy(-1);
        for (ListNode* p = &dummy; l1 != nullptr || l2 != nullptr; p = p->next) {
            int val1 = l1 == nullptr ? INT_MAX : l1->val;
            int val2 = l2 == nullptr ? INT_MAX : l2->val;
            if (val1 <= val2) {
                p->next = l1;
                l1 = l1->next;
            } else {
                p->next = l2;
                l2 = l2->next;
            }
        }
        return dummy.next;
    }
};
```

顶
踩

我的同类文章

高效面试宝典（*15*）

http://blog.csdn.net[2]

- •高效面试之操作系统常考题[3]2014-10-26阅读**870**
- •高效面试之贪心算法[4]2014-10-26阅读**588**
- •高效面试之动态规划DP[5]2014-10-26阅读**2762**
- •高效面试之二叉树[6]2014-10-26阅读**369**
- •高效面试之栈[7]2014-10-26阅读**209**

更多文章[8]

Links

1. javascript:void(0);
2. http://blog.csdn.net/
3. http://blog.csdn.net/cqkxboy168/article/details/40465871
4. http://blog.csdn.net/cqkxboy168/article/details/40465629
5. http://blog.csdn.net/cqkxboy168/article/details/40465389
6. http://blog.csdn.net/cqkxboy168/article/details/40464865
7. http://blog.csdn.net/cqkxboy168/article/details/40464601
8. http://blog.csdn.net/cqkxboy168/article/category/1427428