链表面试题精讲

七月算法　**曹鹏**

2015年4月24日

提纲

- 链表简介
- 面试题总体分析
- 一些例题
  - 例1 链表的插入与（懒）删除
  - 例2 链表翻转
  - 例3 单链表找环及起点和环长度
  - 例4 两个链表找交点
  - 例5 复制带有随机指针的链表
  - 例6 链表partition过程
- 总结

链表简介

- 链表：一个元素和下一个元素靠指针连接(松散)，不能O(1)直接访问到第k个元素
  - 单（向）链表：只能找到下一个节点
  - 双（向）链表：能找到上一个和下一个节点
  - 循环（单、双）链表：首尾相接形成环
- Java : LinkedList
- C++ : STL list
- C : 指针

面试题总体分析

链表的基本操作

- 插入
- 删除
- （分组）翻转
- 排序 Partition、归并
- 复制
- 归并排序
- 找环、起点、长度
- （倒数）第k个节点
- 随机返回一个节点
- 和其他数据结构（二叉树）相互转换

例1 链表的插入与删除

- 例1 在单链表里插入/删除一个节点
  - 插入
    - 哪些指针要修改？前驱的next， 新节点的next
    - 我们要找到插入之前的那个节点
    - 特殊情况: 在head之前插入（包括head == NULL)

```
now->next = head;
head = now;
```

    - 一般情况:在pre后面插入

```
now->next = pre->next;
pre->next = now;
```

例1 续1

- 删除
  - 哪些指针要修改？前驱的next
  - 我们要找到删除之前的那个节点
  - 特殊情况？ 删除head

    temp = head->next;

    delete head;

    head = temp;
  - 一般情况，在pre后面删除

    temp = pre->next;

    pre->next = temp->next;

    delete temp;

例1 续2

- 思考题
  - 双向链表的插入、删除
  - 循环有序链表的插入、删除（建议断开、再连上）
  - “懒”删除
    - 要删除now这个节点(不是最后一个)
    - 把now复制成now->next
      - now->x = now->next->x
    - 删除now->next

例2 单链表翻转

例2 单链表翻转

- 思路：把当前节点拿过来作为已经翻转结果的表头（堆栈类似）

```
ListNode *result = 0;
while (head) {
temp = head->next;   //保存下一个节点
    head->next = result;   //当前节点放到结果的开头
    result = head;          //当前节点的头
    head = temp;      //head指向下一个节点
}
return result;
```

例2 续

思考题

- 翻转部分链表 (Leetcode 92)
  - 如何找到第m个元素和第n个元素
  - 如何处理前面和后面？
    - 保存前面部分最后一个元素
    - 保存后面部分第一个元素
    - 特殊情况？
- 每k个元素翻转一次 (Leetcode 25)
    - 前面翻好的部分（小链表）
    - 要翻转的部分(K个)
    - 后面没处理的部分（小链表）
    - 不足k个怎么办

例3

- 例3 单链表里是否有环？如果有起点是哪里？环长度是多大？（最后一个节点next不是空，而是前面某个节点）(Leetcode 141, 142)
  - 方法1 用一个set存放每个节点地址
    - 注意: set存放的元素必须“有序”，而地址都是“整数”

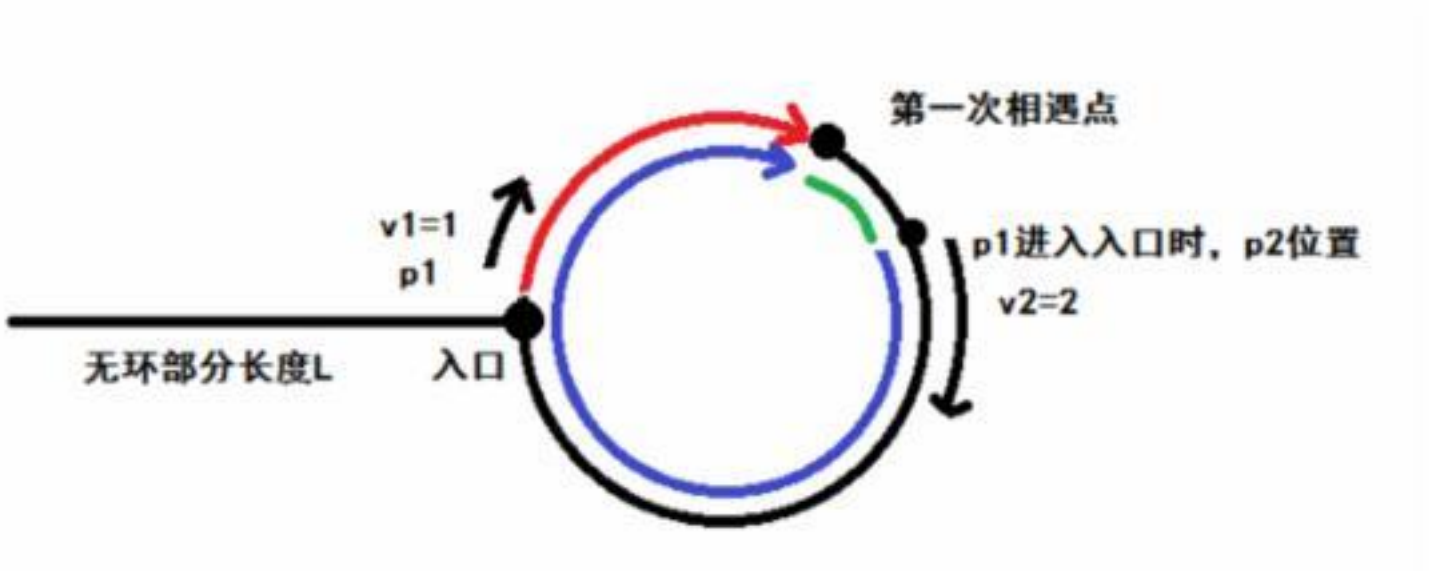
```
set<ListNode*> have;
for (; head; head = head->next) {
        if (have.find(head) != have.end()) return true;
            have.insert(head);
}
return false;
```

例3 续1

- 方法2 不用set?
  - 用两个指针p1和p2, p1每次走一步，p2每次走两步，如果有圈一定会相遇
  - 为什么一定会相遇?
  - 相遇时如何找交点?
  - 一些变量
    - 圈长 n
    - 起点到圈的起点距离a
    - p1到圈起点时，p2在圈中的位置(0<= x < n)

例3 续2

- p1到起点后，经过n - x步相遇（追及问题）
-
-
-
-
-
- 如何找圈长？
  - 相遇后,p2再走一圈并统计长度就是圈长



例3 续3

```
/**
 * 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) {
        ListNode *p1 = head, *p2 = head;
        do {
            if ((p2 == 0) || (p2->next == 0)) {
                return 0;
            }
            p2 = p2->next->next;
            p1 = p1->next;
        } while (p1 != p2);
        for (p1 = head; p1 != p2; p1 = p1->next, p2 = p2->next)
        ;
        return p1;
    }
};
```

例4

- 例4 单向链表找交点 (Leetcode 160)
  - 方法1: set记录一个链表里所有的节点
  - 方法2: 一个链表长x，另外一个链表长y，(x >= y),第一个链表先走x－y步，再一起走……
  - 方法3: 我们把第一个链表首尾相接，连成一个环，使用例3的方法在第二个链表里找圈的起点就是链表的交点（最后别忘记恢复第一个链表——从表头找到表尾,next设置为空）

例4 续

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    int getLength(ListNode *head) {
        int r = 0;
        for (; head; head = head->next, ++r)
        ;
        return r;
    }
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
       int lenA = getLength(headA), lenB = getLength(headB);
       if (lenA >= lenB) {
           for (int i = lenA - lenB; i; --i, headA = headA->next)
           ;
       }
       else {
           for (int i = lenB - lenA; i; --i, headB = headB->next)
           ;
       }
       for (; headA && headB && headA != headB; headA = headA->next, headB = headB->next)
       ;
       return (headA == headB)?headA:0;

    }
};
```

例5

- 例5 一个单链表除了next指针外还有一个random指针随机指向任何一个元素(可能为空)，请复制它 (Leetcode 138)
  - 难点：我们不知道random指针在复制后链表的地址——复制元素地址变了
  - 方法1 map<旧地址，新地址>，先按照普通方法复制链表，再两个链表同时走复制random（旧节点a,新节点a’)

    a’ ->random = map[a->random] （空单独处理）

例5 续1

- 方法2 不用map
  - 插入：每个旧节点后面插入一个自身的“复本”
  - 复制random指针
    - 一个旧节点a的复本是a->next
    - a->random的复本是a->random->next
    - 新节点的random指针a->next->random = a->random->next (空值单独判断)
  - 拆分
    - 旧节点链表是奇数项
    - 新节点链表是偶数项

例5 续2

```
/**
 * Definition for singly-linked list with a random pointer.
 * struct RandomListNode {
 *     int label;
 *     RandomListNode *next, *random;
 *     RandomListNode(int x) : label(x), next(NULL), random(NULL) {}
 * };
 */
class Solution {
public:
    RandomListNode *copyRandomList(RandomListNode *head) {
        if (head == 0) {
            return 0;
        }
        for (RandomListNode *now = head;now;) {
            RandomListNode *copy = new  RandomListNode(now->label);
            copy->next = now->next;
            now->next = copy;
            now = copy->next;
        }
        for (RandomListNode *now = head;now;now = now->next->next) {
            now->next->random = (now->random == 0)?0:now->random->next;
        }
        RandomListNode *h = head->next, *t = h, *tail = head;
        for (;;) {
            tail = tail->next = t->next;
            if (tail == 0) {
                break;
            }
            t = t->next = tail->next;

        }
        return h;
    }
};
```

例6

- 例6 链表partition

链表里存放整数，给定x把比x小的节点放到>=x之前(Leetcode 86)

```
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        ListNode *h1 = 0, *t1 = 0, *h2 = 0, *t2 = 0;
        for (; head; head = head->next) {
            if (head->val < x) {
                if (t1) {
                    t1 = t1->next = head;
                }
                else {
                    h1 = t1 = head;
                }
            }
            else if (t2) {
                t2 = t2->next = head;
            }
            else {
                h2 = t2 = head;
            }
        }
        if (t2) {
            t2->next = 0;
        }
        if (t1) {
            t1->next = h2;
        }
        return h1?h1:h2;
    }
};
```

总结

- 细致——多写代码 多练习
  - 哪些指针要修改
  - 修改前保存（防止链表断掉）
  - 注意空指针
- 特点：可以重新建立表头
  - 翻转（例2）
  - Partition （例6）
  - 注意：第一个元素， 表尾
- 指针：就是int值（地址）