4，链表

2018年4月27日 16:48

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

一些例题

例1 链表的插入与（懒）删除

```
/*
问题：在单链表中插入/删除一个节点

分析：一般链表这种推荐画图。
    1,插入：首先哪些指针需要修改？插入位置之前的节点的next（指向插入节点),插入节点的next（指向之前节点的下一个节点）
    特殊情况：在head之前插入（包括head == NULL)
        now->next = head;
        head = now;
    一般情况：
        now->next = pre->next;
        pre->next = now;


    2,删除：首先哪些指针需要修改？前驱的next
    特殊情况：删除head
        temp = head->next;
        delete head;
        head = temp;
    一般情况：
        temp = pre->next;
        pre->next = temp->next;
        delete temp;


    思考题：双向链表的插入删除
            循环有序的插入删除（建议先断开，再连上）

*/
```

来自 <http://tool.oschina.net/highlight>

例2 链表翻转

```
/*
问题：单链表的反转

分析：维持pre now next三个指针即可

思考题：反转m到n之间的部分，leetcode92
        每k个反转一次 leetcode25
*/


class Solution {
public:
    //二叉链表的翻转，有递归和非递归两种实现
```

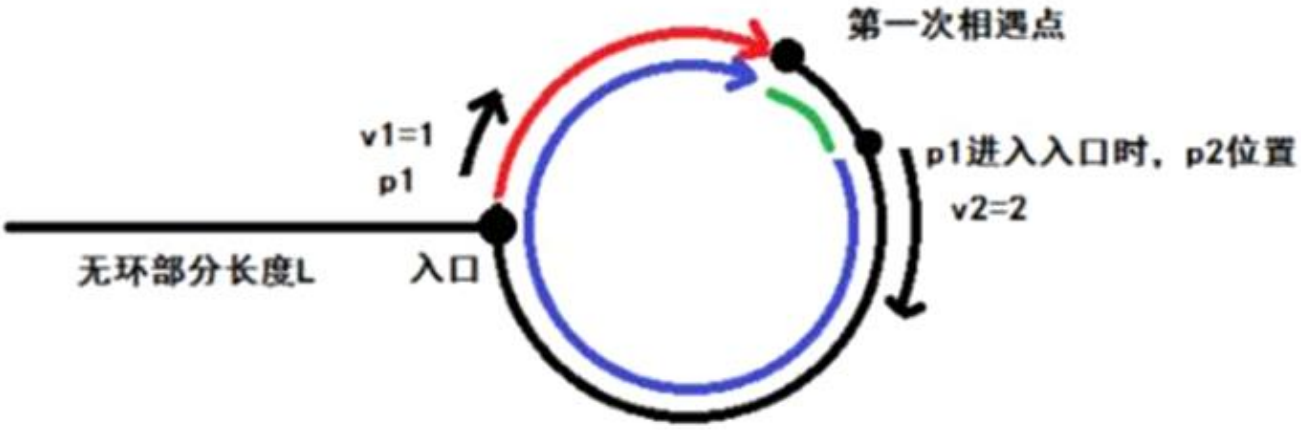
```
    /*
    递归版本：需要维护三个指针，pre，head(当前节点),next
    不断更新三者的值，并将head的next指向pre即可。*/

    ListNode* reverseList(ListNode* head) {
        ListNode* pre = NULL;
        while(head)
        {
            ListNode* next = head->next;
            head->next = pre;
            pre = head;
            head = next;
        }
        return pre;//这里肯定要返回pre，因为最后head = next为空时跳出while。
    }
    /*递归版本
    递归结束条件，递归处理head->next（以当前节点的下一个节点开始的后面部分），reverse
当前节点head和它的next节点。*/
    ListNode* reverseList(ListNode* head) {
       if(!head || !(head->next))
           return head;//递归停止条件
        ListNode *node = reverseList(head->next);
        (head->next)->next = head;
        head->next =NULL;
        return node;
    }

};
```

来自 <http://tool.oschina.net/highlight>

例3 单链表找环及起点和环长度




```
/*https://leetcode.com/problems/linked-list-cycle-ii/description/
问题: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)

分析：方法1：用set存放每个被查找过得节点
    set<listNode*> have;
    for(;head;head = head->next)
    {
        if(have.find(head)!=have.end()) return ture;
```

```
            have.insert(head);
        }
        return false;
```

方法2：但上面的方法存在一定的时间和空间复杂度。
这里我们假设两个指针，p1,p2,p1一次走一步，p2一次走两步。如果有圈一定相遇，为什么呢？建议画图
我们定义一些变量，首先圈长n，起点到圈起点的距离为a，即勺柄的长度，当p1到圈起点时，p2在圈中的位置为x(0<=x<n)
假设相遇时距圈起点为b，那么p1走了a+b，p2走了a+b+k*n = 2(a+b) ,因此有a+b = k×n
如何找圈的起点？
把p1拉回起点，p2从相遇点（b)继续走,这时速率都是1步，a步后，p1到圈起点，p2也刚好到圈起点。
如何找圈长？
相遇后，p2再走一圈并统计长度就是圈长。比如p1不动，p2以1的速率走一圈，与p1相遇即是圈长
*/

```
//141,是否有环
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if(head == NULL) return false;
        ListNode *walker = head;
        ListNode *runner = head;
        while(runner->next!=NULL &&runner->next->next!=NULL)
        {
            walker = walker->next;
            runner = runner->next->next;
            if(walker == runner)
                return true;
        }
        return false;
    }
};
//142,返回环起点和环长。
/**
 * 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) return NULL;//特殊情况
        ListNode *walker = head,*runner = head;
        bool iscycle = false;
        while(runner->next!=NULL && runner->next->next!=NULL)//判环，
```

```
        {
            walker = walker->next;
            runner = runner->next->next;
            if(walker == runner)
            {
                iscycle = true;
                break;
            }
        }
        if(iscycle)//有环，walker重新从起点出发，runner从相遇点出发，步速均为1，再相遇即为环起点。
        {
            for(walker = head;walker!=runner;walker= walker->next,runner = runner->next);
            return walker;
        }
        return NULL;//无环
    }
};
```

来自 <http://tool.oschina.net/highlight>

例4 两个链表找交点

```
/*https://leetcode.com/problems/intersection-of-two-linked-lists/description/
问题：两个链表找交点 单向链表找交点 (Leetcode 160)

分析:可以理解为y字形，一个链表长x，一个链表长y，x>=y.
    方法1：让第一个链表先走x-y步，再一起走，每步判断是否相交即可。
    方法2：如下，先一起走，当一方为空，便指向另一条的头节点，如此，来抵消长度差距。
*/


/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        ListNode *p1=headA,*p2 = headB;
        if(p1 == NULL || p2==NULL ) return NULL;//特殊情况
        while(p1!=NULL&&p2!=NULL&&p1!=p2)//都非空且没相遇
        {
            p1=p1->next;
            p2 = p2->next;
            if(p1 == p2) return p1;//相遇返回节点
            if(p1==NULL) p1 = headB;//为空，p1指向另一条的头节点
            if(p2 == NULL) p2= headA;
        }
        return p1;
    }
};
```

来自 <http://tool.oschina.net/highlight>

例5 复制带有随机指针的链表

/*https://leetcode.com/problems/copy-list-with-random-pointer/
问题：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)

分析：random指针指向随机一个固定的节点，但这个指向是没有规律的。

一个比较好的解法如下：
1,在每个旧节点的后面插入一个自身的副本
2,复制randome指针
一个旧节点a的副本为a->next;
a->random的副本为a->random->next
新节点的randome指针a->next->random = a->random->next;(空值单独判断)
3,拆分：
旧节点链表是奇数项，新节点链表是偶数项。

*/

```
/**
 * 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:
    /*
        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


    */
    RandomListNode *copyRandomList(RandomListNode *head) {
        if(!head) return NULL;//特殊情况
        for(RandomListNode *now = head;now!=NULL;)//复制副本
        {
            RandomListNode *copy = new RandomListNode(now->label);
            copy->next = now->next;
            now->next = copy;
            now = copy->next;
        }
        for(RandomListNode *now = head;now!=NULL;now = now->next->next)//复制random
指针
        {
            now->next->random = (now->random == NULL)?NULL:now->random->next;
        }
        RandomListNode *h = head->next,*t = h,*tail = head;//拆分,h为新复制链表的表
```

```
头，t为新的链表的表尾，tail为旧的链表表尾
        for(;;)
        {
            tail = tail->next = t->next;//每次更新表尾,tail->next = t->next,因为t是
tail的下一项，即副本（这里赋值时先执行右边的）
            if(tail == NULL)
                break;
            t = t->next = tail->next;//tail的下一项是t，
        }
        return h;
    }
};
```

来自 <http://tool.oschina.net/highlight>

例6 链表partition过程

```
/*
问题： 链表里存放整数，给定x把比x小的节点放到x之前  (Leetcode 86)
https://leetcode.com/problems/partition-list/description/
*/
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    //思路，维持两个链表，再合并即可
    ListNode* partition(ListNode* head, int x) {
        ListNode node1(0),node2(0);//两个链表前面的那个节点
        ListNode *p1 = &node1,*p2 = &node2;//指向两个链表的指针
        while(head)
        {
            if(head->val<x)
            {
                p1->next = head;
                p1 = p1->next;
            }
            else
            {
                p2->next = head;
                p2= p2->next;
            }
            head = head->next;
        }
        //将两个链表链接起来
        p2->next = NULL;
        p1->next = node2.next;
        return node1.next;
    }
};
```

来自 <http://tool.oschina.net/highlight>

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）