141. Linked List Cycle

Given a linked list, determine if it has a cycle in it.

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

**S1: Hashing**

Use HashSet, 放入出现的nodes，直到出现已经出现过的/直到最后都没有。

Space: O(n), n:# of nodes, time O(n)

**public** boolean hasCycle(ListNode head) {

Set<ListNode> set = **new** HashSet<>();

**while** (head != **null**) {

**if** (set.contains(head)) **return** **true**;

set.add(head);

head = head.next;

}

**return** **false**;

}

**S2: fast-slow pointer**

用两个不同速度的pointer扫描链表，这样如果是cycled linked list，fast pointer会赶上slow pointer.

public boolean hasCycle(ListNode head) {

if (head == null || head.next == null) return false;

ListNode walker = head;

ListNode runner = head;

while (runner.next.next != null) {

walker = walker.next;

runner = runner.next.next;

if (runner.next == null) return false;

if (walker == runner) return true;

}

return false;

}

142. Linked List Cycle II

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

Note: Do not modify the linked list.

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

**S1.Hashing**

use HashSet

**S2.Fast-slow**

假设环外有m个node，环内有n个nodes，两个node第一次相遇，分别走了m+i, m+n+i步，并且 m + n + i = 2\*(m + i), so m = n - i, 此时fast再走n – i步，就可以到相遇的点，为了保证fast走n-i步，先把slow放回head，两个node用同样速度前进，那么在slow走完m的时候，fast刚好走完n-i, 两者又会相遇，此时的相遇点就是cycle begin点.

public ListNode detectCycle(ListNode head) {

if (head == null || head.next == null) return null;

if (head.next == head) return head;

ListNode slow = head;

ListNode fast = head;

while (fast != null) {

if (fast.next == null)

return null;

slow = slow.next;

fast = fast.next.next;

if (slow == fast)

break;

}

if (fast == null || fast.next == null) return null;

slow = head;

while (slow != fast) {

fast = fast.next;

slow = slow.next;

}

return fast;

}

2. Add Two Numbers

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

**S1**

因为是in reverse order, 所以实际计算的时候，就直接从前往后加就好。

需要注意的：加到最后可能会还有进位1; 两个input可能不一样长

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

ListNode head = new ListNode(0);

ListNode list = head;

int sum = 0;

while (l1 != null || l2 != null) {

if (l1 != null) {

sum += l1.val;

l1 = l1.next;

}

if (l2 != null) {

sum += l2.val;

l2 = l2.next;

}

list.next = new ListNode(sum % 10);

list = list.next;

sum /= 10;

}

if (sum > 0) {

list.next = new ListNode(1);

}

return head.next;

}

445. Add Two Numbers II

you are given two non-empty linked lists representing two non-negative integers. The most significant digit comes first and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

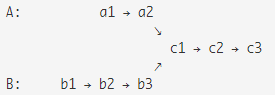
Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 8 -> 0 -> 7

160. Intersection of Two Linked Lists

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists: begin to intersect at node c1.



题解：

思路很简单，先数长度，假设一个比另一个长k，那就先走k step

public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

if (headA == null || headB == null) return null;

int lenA = 0, lenB = 0;

ListNode nodeA = headA, nodeB = headB;

*// 1.get length of both linkedlist:*

while (nodeA != null) {

lenA++;

nodeA = nodeA.next;

}

while (nodeB != null) {

lenB++;

nodeB = nodeB.next;

}

*// 2.for the linkedlist which is k steps longer:work k steps first*

int diff = lenA - lenB;

while (diff > 0) {

headA = headA.next;

diff--;

}

while (diff < 0) {

headB = headB.next;

diff++;

}

while (headA != null) {

if (headA == headB) return headA;

headA = headA.next;

headB = headB.next;

}

return null;

}

Easy-21. Merge Two Sorted Lists

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

两种方法，time O(n), n is length of l1 + l2

**S1. recursion**

+ 不需要新建一个head，

-- 当lists非常长的时候可能stack overflow

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

if (l1 == null) return l2;

if (l2 == null) return l1;

if (l1.val < l2.val) {

l1.next = mergeTwoLists(l1.next, l2);

return l1;

} else {

l2.next = mergeTwoLists(l1, l2.next);

return l2;

}

}

**S2.Iteration**

加入一个固定的head指针，和一个随l1, l2移动的curr指针

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

ListNode head = new ListNode(0); *// stay still*

ListNode curr = head; *// keep moving with l1 & l2*

while (l1 != null && l2 != null) {

if (l1.val < l2.val) {

curr.next = l1;

l1 = l1.next;

} else {

curr.next = l2;

l2 = l2.next;

}

curr = curr.next;

}

if (l1 != null) curr.next = l1;

else curr.next = l2;

return head.next;

}

Easy-23. Merge k Sorted Lists

S1. Recursive merging,

两两合并，直到合并k个lists,共合并logk次。假设共有n个nodes，time O(nlogk)

public ListNode mergeKLists(ListNode[] lists) {

if (lists.length == 0) return null;

if (lists.length == 1) return lists[0];

if (lists.length == 2) return merge2Lists(lists[0], lists[1]);

int mid = lists.length / 2;

return merge2Lists(mergeKLists(Arrays.copyOfRange(lists, 0, mid)), mergeKLists(Arrays.copyOfRange(lists, mid, lists.length)));

}

// 见上一页

public ListNode merge2Lists(ListNode l1, ListNode l2) {// ...}

**S2. Priority Queue**

每次从每个list中取出最小值放入queue，排序是logk，所以时间O(nlogk)

public ListNode mergeKLists(ListNode[] lists) {

if (lists.length == 0) return null;

PriorityQueue<ListNode> queue = new PriorityQueue<ListNode>(lists.length, new ListNodeComparer());

for (ListNode node:lists) {

if (node != null) queue.offer(node);

}

ListNode dummy = new ListNode(0);

ListNode head = dummy;

while (!queue.isEmpty()) {

ListNode cur = queue.poll();

head.next = cur;

head = head.next;

if (cur.next != null) queue.offer(cur.next);

}

return dummy.next;

}

private class ListNodeComparer implements Comparator<ListNode> {

@Override

public int compare(ListNode l1, ListNode l2) {

return Integer.compare(l1.val, l2.val);

}

}

148. Sort List - 149. Insertion Sort List

Sort a linked list in *O* ( *n* log *n* ) time using constant space complexity.

**S1.Merge Sort**

Divide and conquer, 先把list拆成两半，再merge，merge的步骤和上两页的21一样，

此处需要注意一下是如何拆分的，用一个prev node完成。

public ListNode sortList(ListNode head) {

if (head == null || head.next == null) return head;

*// divide into 2 lists*

ListNode fast = head;

ListNode slow = head;

ListNode prev = slow;

while (fast != null && fast.next != null) {

prev = slow;

slow = slow.next;

fast = fast.next.next;

}

prev.next = null; *// cut list into 2 parts*

*// if 1->2->3->4->5, then when fast is 5, slow is 3, middle*

*// if 1->2->3->4->5->6, then when fast is 5, slow is 3, middle/prev half*

*// sort left part and right part*

ListNode l1 = sortList(head);

ListNode l2 = sortList(slow);

*// merge 2 lists*

return merge(l1, l2);

}

206. Reverse Linked List

Reverse a singly linked list.

*// 0(dummy)-1(head, start)-2(then)-3-4*

public ListNode reverseList(ListNode head) {

if (head == null || head.next == null) return head;

ListNode dummy = new ListNode(0);

dummy.next = head;

ListNode start = head; *// stay still, node 1, won't move*

ListNode then = start.next; *// nodes needs to move, 2-3-4*

while (then != null) {

start.next = then.next;

then.next = dummy.next;

dummy.next = then;

then = start.next;

}

return dummy.next;

}

92. Reverse Linked List II

Reverse a linked list from position m to n. Do it in-place and in one-pass.

For example:

Given 1->2->3->4->5->NULL, m = 2 and n = 4,

return 1->4->3->2->5->NULL

**S 同上，只是先到指定位置，再移动指定个数**

public ListNode reverseBetween(ListNode head, int m, int n) {

if (head == null || head.next == null || m == n) return head;

ListNode dummy = new ListNode(0);

dummy.next = head;

ListNode pre = dummy;

for (int i = 1;i < m;i++) {

pre = pre.next;

}

*// 1-(2-3-4)-5 m=2, n=4, here pre=1, start=2,then=3(the node be removed)*

*// first: 1-(3-2-4)-5 swap the position of 2 and 3*

*// second: move 4 to the position before 3*

ListNode start = pre.next;

ListNode then = start.next;

for (int i = m;i < n;i++) {

start.next = then.next;

then.next = pre.next;

pre.next = then;

then = start.next;

}

return dummy.next;

}

24. Swap Nodes in Pairs

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.

**S1 Recursion**

public ListNode swapPairs(ListNode head) {

if (head == null || head.next == null) return head;

ListNode cur = head.next;

head.next = swapPairs(cur.next);

cur.next = head;

return cur;

}

25. Reverse Nodes in k-Group

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

S1. 用Reverse Node in Pairs 的方法+Reverse all nodes 的方法

不确定分类：

138. Copy List with Random Pointer

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null. Return a deep copy of the list.

这道题的time 一定是O(n)，但是space可以从O(n)优化到O(1)

**S1 Map**

用map来存<old node, new node>之间的对应关系，来给new list的每个node设置random pointer

public RandomListNode copyRandomList(RandomListNode head) {

if (head == null) return null;

Map<RandomListNode, RandomListNode> map = new HashMap<RandomListNode, RandomListNode>();

*// 1. create new node and put into map*

RandomListNode cur = head;

while (cur != null) {

map.put(cur, new RandomListNode(cur.label));

cur = cur.next;

}

*// 2. link new nodes together*

cur = head;

while (cur != null) {

map.get(cur).next = map.get(cur.next);

map.get(cur).random = map.get(cur.random);

cur = cur.next;

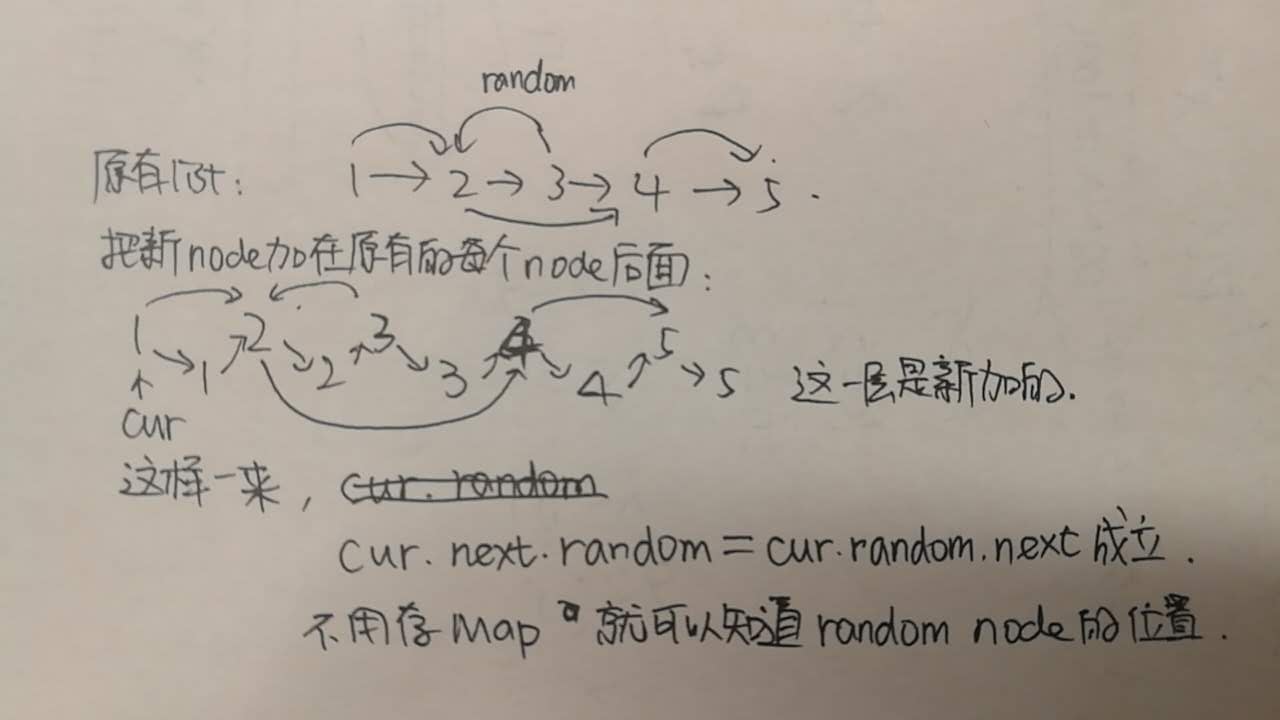
}

return map.get(head);

}

**S2 Optimize space to O(1)**

如果原有list可以改动，则可以优化空间到O(1).



public RandomListNode copyRandomList(RandomListNode head) {

*//if (head == null) return head;*

RandomListNode cur = head;

RandomListNode next = null;

*// 1. copy each node, and put it exactly after each origin node.*

while (cur != null) {

next = cur.next;

RandomListNode copy = new RandomListNode(cur.label);

cur.next = copy;

copy.next = next;

cur = next;

}

*// 2. set random pointer for each copy noe*

cur = head;

while (cur != null) {

if (cur.random != null) {

cur.next.random = cur.random.next;

}

cur = cur.next.next;

}

*// 3.split original list and new list apart*

cur = head;

RandomListNode dummy = new RandomListNode(0);

RandomListNode prevCopy = dummy;

while (cur != null) {

next = cur.next.next;

*// form the new copy list*

prevCopy.next = cur.next;

prevCopy = prevCopy.next;

*// set original list back*

cur.next = next;

cur = cur.next;

}

return dummy.next;

}

109. Convert Sorted List to Binary Search Tree

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST

如果是array会非常好做，直接去中间点为root，再取left middle, right middle作为left/right...

但是linkedlist不能直接access.

**S1. fast, slow pointers**来找中间的点

use 2 nodes, fast and slow, to find the middle node every time.

use a helper function, boundary condition is head == tail，

Time: O(nlogn), since have to traverse sub-list in each recursive call.

public TreeNode sortedListToBST(ListNode head) {

if (head == null) return null;

return toBST(head, null);

}

*//key point: 这里的tail是excluded , 不容易出错*

TreeNode toBST(ListNode head, ListNode tail) {

if (head == tail) return null;

ListNode slow = head;

ListNode fast = head;

while (fast != tail && fast.next != tail) {

fast = fast.next.next;

slow = slow.next;

}

TreeNode mid = new TreeNode(slow.val);

mid.left = toBST(head, slow);

mid.right = toBST(slow.next, tail);

return mid;

}