Chapter 4 LinkedList

https://leetcode.com/tag/linked-list/

Remove Zero Sum Consecutive Nodes from Linked List

Given the head of a linked list, we repeatedly delete consecutive sequences of nodes that sum to 0 until there are no such sequences.

After doing so, return the head of the final linked list. You may return any such answer.

(Note that in the examples below, all sequences are serializations of ListNode objects.)

Example 1:

```
Input: head = [1,2,-3,3,1]
Output: [3,1]
Note: The answer [1,2,1] would also be accepted.
```

Example 2:

```
Input: head = [1,2,3,-3,4]
Output: [1,2,4]
```

Example 3:

```
Input: head = [1,2,3,-3,-2]
Output: [1]
```

使用 cumulative sum 来确定 zero sum sublist

```
public ListNode removeZeroSumSublists(ListNode head) {
    int prefix = 0;
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    Map<Integer, ListNode> seen = new HashMap<>();
    seen.put(0, dummy);
    for (ListNode i = dummy; i != null; i = i.next) {
        prefix += i.val;
        seen.put(prefix, i);
    }
    prefix = 0;
    for (ListNode i = dummy; i != null; i = i.next) {
        prefix += i.val;
        i.next = seen.get(prefix).next;
    }
    return dummy.next;
}
```

Odd Even Linked List
Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example 1:

```
Input: 1->2->3->4->5->NULL
Output: 1->3->5->2->4->NULL
```

Example 2:

```
Input: 2->1->3->5->6->4->7->NULL
Output: 2->3->6->7->1->5->4->NULL
```

Maintain 两个 pointer，一个是 odd，一个是 even 的

```
public ListNode oddEvenList(ListNode head) {
    if (head != null) {
        ListNode odd = head, even = head.next, evenHead = even;

        while (even != null && even.next != null) {
            odd.next = odd.next.next;
            even.next = even.next.next;
            odd = odd.next;
            even = even.next;
        }
        odd.next = evenHead;
    }
    return head;
}
```

Merge k Sorted Lists

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

Example:

```
Input:
[
  1->4->5,
  1->3->4,
  2->6
]
Output: 1->1->2->3->4->4->5->6
```

```
public ListNode mergeKLists(List<ListNode> lists) {
    if (lists==null||lists.size()==0) return null;

    PriorityQueue<ListNode> queue= new
PriorityQueue<ListNode>(lists.size(),new Comparator<ListNode>(){
        @Override
        public int compare(ListNode o1,ListNode o2){
            if (o1.val<o2.val)
                return -1;
            else if (o1.val==o2.val)
                return 0;
            else
                return 1;
        }
    });

    ListNode dummy = new ListNode(0);
    ListNode tail=dummy;

    for (ListNode node:lists)
        if (node!=null)
            queue.add(node);

    while (!queue.isEmpty()){
        tail.next=queue.poll();
        tail=tail.next;

        if (tail.next!=null)
            queue.add(tail.next);
    }
    return dummy.next;
}
```

[Linked List Components](#)

We are given head, the head node of a linked list containing unique integer values. We are also given the list G, a subset of the values in the linked list.

Return the number of connected components in G, where two values are connected if they appear consecutively in the linked list.

Example 1:

```
Input:
head: 0->1->2->3
G = [0, 1, 3]
Output: 2
Explanation:
0 and 1 are connected, so [0, 1] and [3] are the two connected components.
```

Example 2:

```
Input:
head: 0->1->2->3->4
G = [0, 3, 1, 4]
Output: 2
Explanation:
0 and 1 are connected, 3 and 4 are connected, so [0, 1] and [3, 4] are the
two connected components.
```

Note:

- If N is the length of the linked list given by head, 1 <= N <= 10000.
- The value of each node in the linked list will be in the range [0, N - 1].
- 1 <= G.length <= 10000.
- G is a subset of all values in the linked list.

```
public int numComponents(ListNode head, int[] G) {
    Set<Integer> setG = new HashSet<>();
    for (int i: G) setG.add(i);
    int res = 0;
    while (head != null) {
        if (setG.contains(head.val) && (head.next == null ||
!setG.contains(head.next.val))) res++;
```

```
        head = head.next;
    }
    return res;
}
```