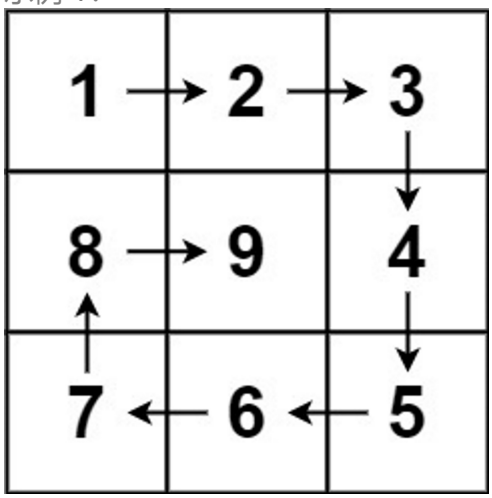
59. 螺旋矩阵 II + 203.移除链表元素

Monday, July 4, 2022 10:08 AM

https://leetcode.cn/problems/spiral-matrix-ii/
中等
给你一个正整数 n ，生成一个包含 1 到 n2 所有元素，且元素按顺时针顺序螺旋排列的 n x n 正方形矩阵 matrix 。

示例 1：



输入：n = 3
输出：[[1,2,3],[8,9,4],[7,6,5]]

解法：

旋转赋值，每次给最外圈赋值，以上图为例：loop为圈数，共n/2圈
第一圈：loop=1，画上面，给（0，0）-（0，n-loop-1）赋值
画右边，给（0,n-loop) - (n-loop-1,n-loop)赋值
再画下边，给(n-loop,n-loop) - (n-loop, loop)赋值
最后画左边，给（n-loop,0) - (loop,0)赋值
再画第二圈...
最后n为奇数时，有中心点，直接赋值

```
class Solution {
    public int[][] generateMatrix(int n) {
        int[][] res=new int[n][n];
        int loop=1; // 循环的次数（画的圈数）共有n/2次
        int start=0;  // 每次循环的开始点(start, start)，第一圈从（0，0），第二圈（1，1）
        int count=1;  // 定义填充数字,从1开始
        int i,j;
        while(loop<=n/2){ // 第一圈为loop=1;
            // 模拟上侧从左到右
            for (j = start; j < n - loop; j++) {
                res[start][j] = count++;
            }
            // 模拟右侧从上到下
            for (i = start; i < n - loop; i++) {
                res[i][j] = count++;
```

```
            }
            // 模拟下侧从右到左
            for (; j >= loop; j--) {
                res[i][j] = count++;
            }
            // 模拟左侧从下到上
            for (; i >= loop; i--) {
                res[i][j] = count++;
            }
            start++;
            loop++;
        }
        // 如果n为奇数的话，需要单独给矩阵最中间的位置赋值,且当n=1时这里直接赋值
        if (n % 2 == 1) {
            res[start][start] = count;
        }
        return res;
    }
}
```

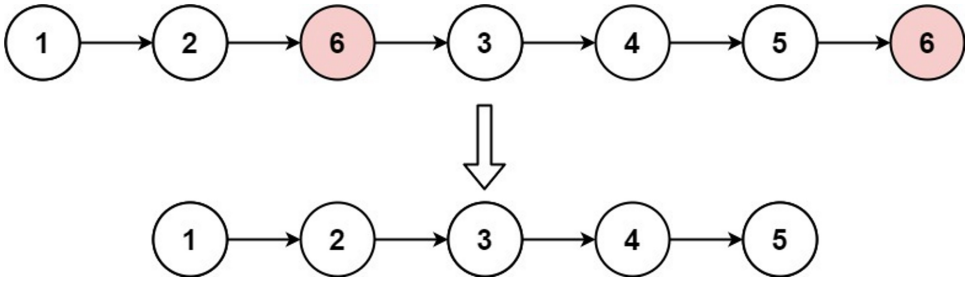
203.移除链表元素

https://leetcode.cn/problems/remove-linked-list-elements/

简单

给你一个链表的头节点 head 和一个整数 val ，请你删除链表中所有满足 Node.val == val 的节点，并返回 新的头节点 。

示例 1：



输入：head = [1,2,6,3,4,5,6], val = 6

输出：[1,2,3,4,5]

```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode res = new ListNode(0); // 新建空节点
        res.next = head; //空节点的next直线head
        ListNode temp = res; // 临时节点指向res，成为代理节点
        while (temp.next != null) { // temp.next 即是原head链表了
            if (temp.next.val == val) {
                temp.next = temp.next.next; //相等跳过该节点
```

```
            } else {
                temp = temp.next; //不等直接后移
            }
        }
        return res.next; //返回的是空头节点后一个
    }
}
```