Leetcode 题解 - 链表

链表删除操作

树是一种递归结构，很多树的问题可以使用递归（深度优先遍历DFS和广度优先遍历BFS）来处理。

1. 移除链表元素

203. 移除链表元素 (简单)

Leetcode / 力扣

虚拟节点统一删除逻辑

```
var removeElements = function(head, val) {
  // 不确定head节点会不会被删除，所以建一个虚拟头节点
  const dummyNode = new ListNode(-1);
  dummyNode.next = head;

  let prev = dummyNode, cur = head;
  while (cur) {
    if (cur.val == val) {
      prev.next = cur.next;
      cur.next = null;
      cur = prev.next; // 更新cur指针
    } else {
      prev = cur;
      cur = cur.next; // 前后指针都向前移动一位
    }
  }
  return dummyNode.next;
};
```

2. 删除链表中的节点

237. 删除链表中的节点 (简单)

Leetcode / 力扣

思路：把删除的那个点的值用后一个值来替代，以至于后面所有值都向前移动一个单位，就相当于删除最后一个节点了，所以需要 prev.next -> null

```
var deleteNode = function(node) {
  let prev = null; // 用于保存要删除节点的前一个节点
  while (node) {
    const next = node.next;
    if (next) {
      node.val = next.val;
    } else {
      prev.next = null;
    }
    prev = node; // pre 向后移动一位
    node = node.next; // node 向后移动一位
  }
};
```

3. 删除排序链表中的重复元素

83. 删除排序链表中的重复元素 (简单)

Leetcode / 力扣

给定一个已排序的链表的头 head ， 删除所有重复的元素，使每个元素只出现一次 。返回 已排序的链表 。

```
var deleteDuplicates = function(head) {
  if (!head || !head.next) return head;

  let prev = head, curr = head.next;
  while (curr) {
      if (curr.val == prev.val) {
          prev.next = curr.next;
          curr.next = null;
      } else {
          prev = curr;
      }
      curr = prev.next;
  }
  return head;
};
```

4. 删除排序链表中的重复元素 II

82. 删除排序链表中的重复元素 II (中等)

Leetcode / 力扣

给定一个已排序的链表的头 head，删除所有重复的元素，使每个元素只出现一次。返回 已排序的链表。

思路：遍历有序的链表(有序，只需要遍历一次)，删除重复的元素

步骤：
1. 新建虚拟头节点，防止链表头部节点元素被删除
2. 如果cur.next 与 cur.next.next元素相同，应该删除它，及后面所有重复的元素 记录重复的元素为 x，依次寻找
3. 如果cur.next 与 cur.next.next 元素不相同，说明只有一个元素 cur.next的节点不重复

```
var deleteDuplicates = function (head) {
  if (!head || !head.next) return head;

  const dumyNode = new ListNode(-1);
  dumyNode.next = head;
  let cur = dumyNode;
  while (cur.next && cur.next.next) {
    if (cur.next.val === cur.next.next.val) {
      const x = cur.next.val;
      while (cur.next && cur.next.val === x) {
        // 删除 cur.next 节点
        cur.next = cur.next.next;
      }
    } else {
      cur = cur.next;
    }
  }

  return dumyNode.next;
};
```

5. 删除链表的倒数第 N 个结点

19. 删除链表的倒数第 N 个结点 (中等)

Leetcode / 力扣

思路：未知具体节点时，采用快慢指针法

1. 快指针先走n个单位，然后慢指针和快指针同时向前移动一个单位
2. 慢指针指向的下一节点，即为要删除的节点

注意：fast.next 不存在的情况，应及时返回头节点

```

```

层次遍历

1. 二叉树的层序遍历

102. 二叉树的层序遍历 (中等)

Leetcode / 力扣

```
    3
   / \
  9   20
      / \
     15  7
输入：root = [3,9,20,null,null,15,7]
输出：[[3],[9,20],[15,7]]
```

思路：用队列保存每一层的节点，当处理每一层节点时，应该先从队列的队头拉出来，放进当前层级的结果集， 然后找它下一层的左右节点，有的话直接放进队列，循环往复就能得到结果

```
var levelOrder = function(root) {
  if (!root) return [];

  // 队列，先进先出
  const res = [], queue = [];
  queue.push(root);
  while (queue.length) {
    const levelNodes = []; // 新建一个存储当前层结点的数组
    const size = queue.length; // 每轮循环遍历处理一层的节点
    for (let i = 0; i < size; i++) {
      const cur = queue.shift();
      levelNodes.push(cur.val);
      // 将遍历处理的节点的左右节点入队，等待后续的处理
      if (cur.left) queue.push(cur.left);
      if (cur.right) queue.push(cur.right);
    }
    res.push(levelNodes)
  }
  return res;
};
```