【第三十七周】金融系统中的 RSA 算法 (一)

1、剑指 Offer II 029. 排序的循环链表

1. 试图在链表中找到相邻的两个节点，如果这两个节点的前一个节点的值比待插入的值小并且后一个节点的值比待插入的值大，那么就将新节点插入这两个节点之间
2. 如果找不到符合条件的两个节点，也就是待插入的值大于链表中已有的最大值或小于已有的最小值，那么新的节点将被插入值最大的节点和值最小的节点

```
/**
 * @param {Node} head
 * @param {number} insertVal
 * @return {Node}
 */
var insert = function(head, insertVal) {
  let node = new Node(insertVal);
  // 当是空链表的情况
  if (head == null) {
    head = node;
    head.next = head;
  } else if (head.next == head) {
    // 当整个链表只有一个节点的情况;
    head.next = node;
    node.next = head;
  } else {
    let cur = head;
    let next = head.next;
    let biggest = head;
    // 试图找到相邻的两个节点cur和next,使得cur的值小于或等于待插入的值且next的值大于或等于待插入的值
    while (!(cur.val <= node.val && next.val >= node.val)&&next!=head) {
```

```
      cur = next;
      next = next.next;
      if (cur.val >= biggest.val) {
        biggest = cur;
      }
    }
    // 如果找到了就将新节点插入它们之间
    if (cur.val <= node.val && next.val >= node.val) {
      cur.next = node;
      node.next = next;
    } else {
      // 如果没有找到符合条件的两个节点，将新的节点插入值最大的节点biggest后面
      node.next = biggest.next;
      biggest.next = node;
    }
  }
  return head;
};
```

2、229. 求众数 II

1. 我们用哈希统计数组中每个元素出现的次数，设数组的长度为 n，返回所有统计次数超过 n/3的元素。

```
/**
 * @param {number[]} nums
 * @return {number[]}
 */
var majorityElement = function(nums) {
    const cnt = new Map();

    for (let i = 0; i < nums.length; i++) {
        if (cnt.has(nums[i])) {
            cnt.set(nums[i], cnt.get(nums[i]) + 1);
        } else {
            cnt.set(nums[i], 1);
        }
    }
    const ans = [];
    for (const x of cnt.keys()) {
        if (cnt.get(x) > Math.floor(nums.length / 3)) {
            ans.push(x);
        }
    }
```

```
    return ans;
};
```

3、1819. 序列中不同最大公约数的数目

1. 力扣上大多数题解都是法一。因为值域是2e5，又是和公约数有关，所以肯定会考虑枚举因数。这里就考虑枚举i，并判定是否存在一个子序列的gcd是i。
2. 开桶c[]记录每个数出现次数，枚举i的所有倍数j*i，则sum(c[j*i])就找到了数组里i的倍数构成的集合。对于每个合法j，求它们的gcd。若求得gcd为1，则说明该集合的gcd就是i，则i对答案有1的贡献；否则i对答案无贡献。
3. 时间复杂度O(n*log)。

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var countDifferentSubsequenceGCDs = function(a) {
  let gcd = (x, y) => !y ? x : gcd(y, x % y)
  let mx = Math.max(...a)
  let c = new Array(mx + 1).fill(0)
  for (let v of a) c[v]++
  let ans = 0
  for (let i = 1; i <= mx; ++i) {
    let g = 0
    for (let j = 1; j * i <= mx; ++j) {
      if (!c[j * i]) continue
      g = gcd(g, j)
      if (g === 1) break
    }
    ans += g === 1
  }
  return ans
};
```

4、2001. 可互换矩形的组数

1. 使用map存储每个数组中两个数相除的结果，其和实际上就是map中每个数出现的次数乘以次数减一的结果，再除以2，并对结果依次累计，便是最终的结果。

```
/**
 * @param {number[][]} rectangles
 * @return {number}
 */
var interchangeableRectangles = function(rectangles) {
    let map = new Map();

    for(let el of rectangles) {
        let tmp = el[0] / el[1]
        if(!map.get(tmp)) {
            map.set(tmp, 1)
        } else {
            map.set(tmp, map.get(tmp) + 1)
        }
    }

    let ans= 0;

    for(let ele of map) {
        ans += (ele[1] * (ele[1] - 1)) / 2
    }
    return ans
};
```

5、1004. 最大连续1的个数 III

1. 双指针从索引 0 出发，维护一个满足条件的窗口（可行解），通过扩张窗口优化这个可行解。
2. 优化到条件被破坏，转而收缩窗口，丢弃元素，让条件重新满足（新的可行解）
3. 然后继续优化这个新的可行解..... 重复上面两步
4. 每次让窗口长度挑战最大纪录，遍历结束，最大纪录就有了

```
var longestOnes = function(A, K) {
    let i = 0, j = 0; // 双指针 维护窗口的左右端
    let maxLen = 0;   // 窗口最长纪录
    let zero = 0;     // 窗口中0的个数

    while (j < A.length) { // 窗口右端越界了就结束
        if (A[j] == 0) {   // 新纳入的是0，更新zero
            zero++;        // 待会要检测0有没有爆表
        }
        while (zero > K) {   // 0爆表了，左端右移，丢弃元素，直到zero重新等于k
            if (A[i] == 0) { // 如果丢弃是0，更新zero
                zero--;
            }
            i++; // 丢弃左端的元素，收缩窗口，为了重新满足条件
```

```
        }

        j++; // 此时窗口是可行解，扩张窗口，优化可行解

        if (j - i > maxLen) { // 如果破了最大纪录，更新它
            maxLen = j - i;
        }
    }

    return maxLen;
};
```