【第三十九课】金融系统中的 RSA 算法 (三)

1、2119. 反转两次的数字

1. num为0，肯定满足，返回true
2. num结尾有0，肯定不满足，返回false
3. 其余情况都是true

```
/**
 * @param {number} num
 * @return {boolean}
 */
var isSameAfterReversals = function(num) {
    if (num === 0) return true;
    if (`${num}`.endsWith('0')) return false;
    return true;
};
```

2、2130. 链表最大孪生和

1. 快慢指针找到链表的中间位置
2. 反转后半部分链表
3. 从两端向中间遍历，找最大值

```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} head
 * @return {number}
 */
var pairSum = function(head) {
  // 快慢指针找到链表中间位置
  let slow = head, fast = head.next
  while(fast.next) {
    slow = slow.next
```

```
    fast = fast.next.next
  }
  // 反转后半部分链表
  let cur = slow, pre = null
  while(cur) {
    const tmp = cur.next
    cur.next = pre
    pre = cur
    cur = tmp
  }
  // 从两端向中间遍历 找最大值
  let p1 = pre, p2 = head, ans = 0
  while(p1 && p2) {
    ans = Math.max(ans, p1.val + p2.val)
    p1 = p1.next
    p2 = p2.next
  }
  return ans
};
```

3、2104. 子数组范围和

1. 子数组是数组中一个**连续 非空** 的元素序列。
2. 这里采用滑动窗口的策略，不断扩大窗口的大小，每扩大一个元素，就维护一下窗口中的最大值和最小值，然后计算中间结果。
3. 就是两个指针暴力枚举

```
 /**
 * @param {number[]} nums
 * @return {number}
 */
var subArrayRanges = function(nums) {
    const n = nums.length;
    let res = 0;
    for (let i = 0; i < n; i ++)
    {
        let min_ = nums[i];
        let max_ = nums[i];
        for (let j = i + 1; j < n; j ++)
        {
            min_ = Math.min(min_, nums[j]);
            max_ = Math.max(max_, nums[j]);
            res += (max_ - min_);
        }
    }
```

```
    return res;
};
```

4、2139. 得到目标值的最少行动次数

1. 使用贪心算法倒序处理，加倍变成减半，递增变成递减
2. 先消耗掉所有减半次数 maxDoubles，消耗过程中如果是偶数则减半，如果是奇数则递减，每次消耗次数 count 都加1
3. 剩余的操作只能是递减，需要的操作次数为剩余整数减1即 target - 1
4. 最后返回 count + target - 1

```
/**
 * @param {number} target
 * @param {number} maxDoubles
 * @return {number}
 */
var minMoves = function (target, maxDoubles) {
  let count = 0;
  while (target > 1 && maxDoubles && ++count) {
    if (target % 2 === 0) maxDoubles--, (target /= 2);
    else target -= 1;
  }
  return count + target - 1;
};
```

5、2136. 全部开花的最早一天

1. 贪心的证明比较难, 其实只需要想明白下面这一点,
2. 不论以任何顺序种下，播种花的总时间总为 sum{plantTime}
3. 所以接下来，将花的生长天数逆序排序
4. 不需要对播种天数进行排序, 生长天数相同的花, 无论先播种哪个结果都一样
5. 生长天数最长的花优先播种

```
var earliestFullBloom = function(plantTime, growTime) {
    const len = plantTime.length;
    let list = []
    let resutl = 0
    for(let i = 0 ; i < len ; i++){
        const x = plantTime[i]// 播种天数
        const y = growTime[i]// 开花天数
       list.push([y,x])
```

```
    }
    list.sort((a,b)=>b[0]- a[0]);

    // 开花期越长，越先种植;
    let day = list[0][1];
    // 花期
    let current = [day + list[0][0]]
    for(let i = 1 ; i < len ; i++){
        // 种植时间
        day+= list[i][1];
        current.push( day+list[i][0])
    }
    const max = Math.max(...current)
    return max

};
```

6、2134. 最少交换次数来组合所有的 1 II

1. 统计数组中 1 的数量 oneCnt;
2. 开一个宽度为 oneCnt 的窗口， 统计窗口里面 0 的数量 zeroCnt（0的个数即为所有1数字相邻需要交换的次数）；
3. 遍历一遍数组的窗口，求出最少需要交换的次数即为此题的解；

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var minSwaps = function(nums) {
        let n = nums.length;
        let ans = n, oneCnt = 0;
        // 统计数组中 1 的数量
        for (let num of nums) {
            if (num == 1) oneCnt++;
        }
        // 开一个宽度为 oneCnt 的窗口， 统计窗口里面 0 的数量 zeroCnt
        // 最少的 zeroCnt 即为此题答案
        for (let i = 0, zeroCnt = 0; i < 2 * n; i++) {
            if (i >= oneCnt) {
                ans = Math.min(ans, zeroCnt);

                // 右移窗口时，被移除的数字是 0 则 zeroCnt--
                if (nums[(i - oneCnt) % n] == 0) {
                    zeroCnt--;
                }
```

```
            }

            if (nums[i % n] == 0) zeroCnt++;
        }
        return ans;
}
```