【第三十六周】有趣的莫比乌斯反演

1、941. 有效的山脉数组

1. 直接 if-else 纯暴力判断每一个点的情况：遍历数组，判断每一个点的情况，判断当前的点和后一个点的大小关系，决定当前点是否满足山脉数组的要求，直到遍历到倒数第二个点为止，并且如果是倒数第二个点了，需要做尾部判断和处理。

```
/**
 * @param {number[]} A
 * @return {boolean}
 */
var validMountainArray = function(A) {
    if (A.length < 2) {
        return false
    }
    let isUp = true // 默认开始为上升阶段
    for (let i = 0; i < A.length - 1; i++) {
        let item = A[i]
        if (item < A[i + 1]) { // 如果当前到下一个值是上升阶段
            if (isUp) { // 如果之前是上升阶段，则继续遍历
                if (i === A.length - 2) { // 处理遍历到最后的情况
                    return false // 因为最后还是上升阶段，不满足山脉数组要求，返回
false
                } else {
                    continue
                }
            } else {
                return false // 如果之前是下降阶段，因为题目一整个山脉是一个数组，不能继
续有上升阶段，所以直接返回 false
            }
        } else if (item > A[i + 1]) { // 如果当前到下一个值是下降阶段
            if (!isUp) { // 如果之前是下降阶段，则继续遍历
                continue
            } else { // 之前是上升阶段，现在转换为下降阶段
                if (i === 0) { // 如果一来就变成下降阶段
                    return false // 说明山脉一开始没有上升阶段，直接返回false
                }
                isUp = false // 换方向
            }
        } else { // 当前遍历值等于下一个值，不满足条件直接返回 false
            return false
        }
```

```
    }
    return true
};
```

2、289. 生命游戏

1. 首先根据题目要求不难知道，其实就是数一数每个数字周围的活细胞数从而得出新的值
2. 由于复活和死亡是同时发生的，这意味着我们需要遍历完数组，然后得出每一个细胞的状态保存起来，然后一次性更新
3. 那么这道题目的难点就在于，如何保存计算出来的这个状态结果，因为结果覆盖了原数字，会导致下一个细胞的状态计算出问题
4. 那么我们可以想到原地修改数组的一种方式：取负值，计算的时候拿绝对值来判断，但是死细胞对应的数字0没法取负，那么我们的思路就捋顺了
5. 我们先遍历一遍把所有数字加一，重新定义状态：1为死细胞，2为活细胞
6. 这样我们再次遍历数组，根据数字的绝对值来计算活细胞数量，如果死亡或者复活就把值设为负数，这样就不会影响到下一次计算

   最后再遍历数组，把所有的结果计算出来即可（设为0或1）

```
/**
* @param {number[][]} board
* @return {void} Do not return anything, modify board in-place instead.
*/
var gameOfLife = function(board) {
    const m = board.length,
        n = board[0].length;

    //遍历加一，用于取负方便
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            board[i][j]++;
        }
    }

    //遍历第二遍计算细胞存活状态
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            changeStatus(board, i, j);
        }
    }

    //遍历第三遍更新细胞存活状态
```

```
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {

            if (board[i][j] === 1 || board[i][j] === -2) {
                //把结果为死细胞的位置置为0
                board[i][j] = 0;
            } else {
                //否则都是活细胞
                board[i][j] = 1;
            }
        }
    }

};

var changeStatus = function(board, r, c) {
    const m = board.length,
        n = board[0].length;

    //活细胞数量
    let num = 0;
    for (let i = Math.max(0, r - 1); i <= Math.min(r + 1, m - 1); i++) {
        for (let j = Math.max(0, c - 1); j <= Math.min(c + 1, n - 1); j++) {
            //记录不包括自己在内的，周围的活细胞数量
            if ((i !== r || j !== c) && Math.abs(board[i][j]) === 2) {
                num++;
            }
        }
    }

    //改变细胞状态
    if ((board[r][c] === 2 && (num < 2 || num > 3)) ||
        (board[r][c] === 1 && num === 3)) {
        //活细胞死亡或者死细胞复活，反转正负
        board[r][c] = -board[r][c];
    }
};
```

3、754. 到达终点数字

1. 数学问题，s = 1 + 2 + 3 + 4 .... + k; 根据添加 + 和 - 计算和。所有target为正数还是负数都一样，所以根据绝对值计算。
2. 1、找到 k , 其中 S(k) >= target; S(k) = 1 + 2 ... + k;
3. 2、如果S(k) === target，没啥说的，step = k;

4. 3.1、如果 (S(k) - target) % 2 === 0; 差为偶数，只要其中一个数字转为 (负数) 即可;
5. 比如 target = 4; S(3) = 1 + 2 + 3 = 6 ; 把 1 转为 -1 即可。即 -1 + 2 + 3 = 4； 因为 n * 2 为偶数
6. 3.2、如果 (S(k) - target) % 2 !== 0; 差为奇数。需要 S(k) 继续往后加直到差为偶数。
7. 如果k为偶数 +1 即可;
8. 如果k为奇数 +2 即可;
9. 例如：S(3) = 1 + 2 + 3 = 6; target = 5; S(3) - target = 1; 差为奇数， k = 3 奇数； 1 + 4 = 5 还是奇数； 1 + 4 + 5 = 10 才为偶数。
10. 例如：S(4) = 1 + 2 + 3 + 4 = 10; target = 9; S(4) - target = 1; 差为奇数， k = 4 偶数； 1 + 5 = 6 即为偶数。

```
/**
 * @param {number} target
 * @return {number}
 */
var reachNumber = function(target) {
  if(target < 0) target = -target;
  let sum = 0;
  let step = 0;
  while(true) {
    sum = (step * step + step) / 2;
    if(sum >= target) break;
    step++;
  }
  const n = sum - target;
  if(n === 0 || n % 2 === 0) {
    return step;
  }
  // 根据奇偶数来判断。
  if (step % 2 === 0) return step + 1;
  return step + 2;
};
```

4、132. 分割回文串 II

1. *dp*[*i*] ： 索引 0 到 i 的子串 [0,i] 的最小分割数，题目求：dp[n-1] , n 为字符串 s 的长度
2. 如果 [0,i] 就是回文串，不用切割，此时 dp[i] = 0
3. *dp*[*i*] 对应的子串长度为 i+1，最多能被分割 i 次
4. 所以我们初始化 dp[i] = i ， 包含了 dp[0] = 0
5. 我们尝试将子问题拆成规模小一点的子问题，找到它们之间的联系。
6. dp[i] 表示 [0,i] 的最小分割数，我们用指针 j 去切分一下 [0,i] ， 切一个规模小一点的 dp 子问题出来。

7. 分成了两部分：[0,j] 和 [j+1,i] ，其中 [0,j] 的最小分割数是 dp[j] ，它相对于 dp[i] 是计算过的状态，我们要找出 dp[i] 和 dp[j] 的递推关系。
8. 对于 [j+1,i] ，如果它是回文串，就有递推关系：dp[i] = dp[j] + 1
9. 因为 j 指针是在扫 [0,i] ，j 在变，它切的 [j+1,i] 如果多次是回文串，dp[i] 取最小的 dp[j]+1 就好
10. 两次 DP：
11. 因为我们需要判断 [j+1,i] 子串是否回文， 又因为用的 dp 二维数组存放每个 [i,j] 子串是否回文。

```
/**
 * @param {string} s
 * @return {number}
 */
// 两次动态规划
var minCut = function(s) {
    const n = s.length
    const g = new Array(n).fill(0).map(() => new Array(n).fill(true))
    // 第一次动态规划 数据预处理 求出字符串 所有子串是否为回文串 true/false
    for (let i = n - 1; i >= 0; --i) {
        for (let j = i + 1; j < n; j++) {
            g[i][j] = (s[i] === s[j]) && g[i + 1][j - 1]
        }
    }

    const f = new Array(n).fill(Number.MAX_SAFE_INTEGER);
    // 第二次动态规划  f[i]为 0 ~ i分割回文串的最小分割次数
    // 如果 0 ~ i 本身为回文串 则分割次数为0
    for (let i = 0; i < n; ++i) {
        if (g[0][i]) {
            f[i] = 0;
        } else {
            for (let j = 0; j < i; ++j) {
                if (g[j + 1][i]) {
                    f[i] = Math.min(f[i], f[j] + 1);
                }
            }
        }
    }
    return f[n - 1];
};
```

5、1155. 掷骰子的N种方法

1. 状态：dp[i][j] 代表 扔 i 个骰子和为 j；
2. 方程：dp[i][j] 与 dp[i - 1] 的关系是什么呢？ 第 i 次我投了 k (1 <= k <= f)，那么前 i - 1 次 和 为 j - k，对应 dp[i - 1][j - k]；

3. 于是有最终方程：dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j - 2] + ... + dp[i - 1][j - f]
4. 边界条件： dp[1][k] = 1 (1<= k <= min(target, f))

```
/**
 * @param {number} d
 * @param {number} f
 * @param {number} target
 * @return {number}
 */
var numRollsToTarget = function(d, f, target) {
    const dp = Array.from({ length: d + 1 }, () => Array(target + 1).fill(0))
    for(let i = 1; i <= f; i++) dp[1][i] = 1
    for(let i = 2; i <= d; i++) {
        for(let j = i; j <= target; j++) {
            for(let k = 1; k < j && k <= f; k++) {
                dp[i][j] = (dp[i][j] + dp[i - 1][j - k]) % (1e9+7)
            }
        }
    }
    return dp[d][target]
};
```

6、1147. 段式回文

1. 左右两边同时遍历，找到相同的段时计数+2即可

```
 /**
 * @param {string} text
 * @return {number}
 */
var longestDecomposition = function(text) {
    let i=0,j=text.length-1;
    let word1="",word2="";
    let ans=0;
    while(i<j){
        word1+=text[i++];
        word2=text[j--]+word2;
        if(word1===word2){
            ans+=2;
            word1=word2=""
        }
    }
    //如果i===j表示这段字符串为计数，最中间必定独自为一段
    //如果word1.length>0最后word1+word2必定独自为一段
    return ans+(word1.length>0 || i===j ?1:0);
```

```
};
```


开课吧