【第十七课】单调队列及经典问题

彩蛋题目：开课吧OJ-127

862. 和至少为 K 的最短子数组

滑动窗口

1.创建一个数组，其中P[i] = A[0]+A[1]+ ...+ A[i]（如果A[i]没有负数可以保证P[i+1]>P[i]）则本题转换为P[y]-P[x]>=k且y-x最小的值

2.使用双端队列保存滑动窗口值（js可以直接使用数组），每次循环在队列尾添加本次循环的下标j，记为滑动窗口末尾值，为了保证P[j+1]>P[j],因此while(queue.length!=0 && P[queue[queue.length-1]]>=P[j]){queue.pop()}，当queue.length!=0 && P[j]-P[queue[0]]>=K**时判断最新的滑动窗口初始值**

```
var shortestSubarray = function(A, K) {
    let P = new Array(A.length+1).fill(0)
    for(let i=0;i<A.length;i++){
        P[i+1]= P[i]+A[i]
    }
    let queue=[],min=A.length+1;
    for(let j=0;j<P.length;j++){
        // 上次的和大于本次的和，即P[j-1]>P[j],则不存取本次的j
        while(queue.length!=0 && P[queue[queue.length-1]] >=P[j]){
            queue.pop()
        }
        while(queue.length!=0 && P[j]-P[queue[0]]>=K){
            // 当本次的P[j]>P[滑动窗口初始值]，则取最小长度
            min = Math.min(j-queue[0],min)
            // 并删除滑动窗口初始值，而后重新push进当前j，则滑动窗口上次结束值为初始值，当前j为滑动窗口结束值
            queue.shift()
        }
        queue.push(j)
    }
    return min<A.length+1?min:-1
};
```

1760. 袋子里最少数目的球

二分

1、注意到题目数据范围10^9，可以想到普通的模拟一定会超时，因此采用二分查找计算结果。

2、二分的目标为最终的最大值结果，判断条件为该最大值结果是否能通过maxOperations次操作得到。

3、注意返回结果为二分上界，因为不能整除的情况需要向上进位。

```
var minimumSize = function(nums, maxOperations) {
    let l = 0, r = 0;
    // 计算二分查找上边界r
    for(let n of nums) {
        r = Math.max(n, r);
    }
    // 注意二分边界条件
    while(l + 1 < r) {
        // JS语言特性：注意Math.floor向下取整
        let mid = Math.floor(l + (r - l) / 2), tmp = 0;
        for(let n of nums) {
            tmp += Math.floor((n - 1) / mid);
        }
        // 当前没有用完操作次数，说明还可以进一步降低最终的最小取值，向下调整上边界
        if(tmp <= maxOperations) {
            r = mid;
        }
        // 当前用完了操作次数，说明当前最小取值无法满足条件，向上调整下边界
        else {
            l = mid;
        }
    }
    // 注意最终返回上边界的值
    return r;
};
```

46. 全排列

回溯

1、用递归模拟出所有情况。

2、遇到包含重复元素的情况，就回溯。

3、收集所有到达递归终点的情况，并返回。

```
var permute = function(nums) {
    const res = [];
    backtrack(nums, []);
    return res;
    function backtrack(nums, track) {
        if (track.length === nums.length) {
            res.push(track);
            return;
        }
        for (let i = 0; i < nums.length; i++) {
            if (track.includes(nums[i])) { continue; };
            track.push(nums[i]);
            const newTrack = [...track];
            backtrack(nums, newTrack);
            track.pop();

        }
    }
};
```

513. 找树左下角的值

1、我们使用前序遍历，优先进行左边搜索，判断当前是否是最大深度，当前结点是否是最左边的结点。

2、我们可以设置2个全局变量，一个记录最大深度，一个记录当前深度。当结点是叶子结点，且其所处深度比已记录的最大深度大时，我们就更新最左值和最大深度值。

3、同深度下只会进行一次值的更新，由于是前序遍历，这唯一一次更新的最左值就是此深度下最左边的值。

4、这样递归完成后，我们就已经找到这颗树最左下角的值了。

```
var findBottomLeftValue = function(root) {
    // 记录最大深度，初值设置为负无穷，方便比较
    let maxLevel = -Infinity,
    // 当前深度
    curLevel = 0,
    // 最左值
    maxLeftVal = 0
    // 前序遍历
    let preOrderTraversal = function(node){
        // 如果结点不存在则返回
        if(!node) return
        // 当前深度递增
        curLevel++
        // 当结点是叶子结点，且当前深度最大时，它便是树最左下角的结点。
        // 前序遍历优先搜索左边的值，同深度下，最左边的结点最先被搜索到
        // 同深度下，此判断语句内的代码只会被执行一次
        if(curLevel > maxLevel && !node.left && !node.right){
            // 记录最大深度
            maxLevel = curLevel
            // 记录最左值
            maxLeftVal = node.val
        }
        // 遍历左子树
        node.left && preOrderTraversal(node.left)
        // 遍历右子树
        node.right && preOrderTraversal(node.right)
        // 回溯，深度递减
        curLevel--
    }
    // 从根结点开始向下遍历
    preOrderTraversal(root)
    return maxLeftVal
};
```

135. 分发糖果

我们遍历该数组两次，处理出每一个学生分别满足左规则或右规则时，最少需要被分得的糖果数量。每个人最终分得的糖果数量即为这两个数量的最大值。

```
var candy = function(ratings) {
    const n = ratings.length;
    const left = new Array(n).fill(0);
    for (let i = 0; i < n; i++) {
        if (i > 0 && ratings[i] > ratings[i - 1]) {
            left[i] = left[i - 1] + 1;
        } else {
            left[i] = 1;
        }
    }

    let right = 0, ret = 0;
    for (let i = n - 1; i > -1; i--) {
        if (i < n - 1 && ratings[i] > ratings[i + 1]) {
            right++;
        } else {
            right = 1;
        }
        ret += Math.max(left[i], right);
    }
    return ret;
};
```

43. 字符串相乘

1.0乘以任何数 = 0
2.两数相乘，乘积的长度一定 <= 两数长度之和
3.被乘数的一位 与 乘数的每一位相乘，乘积不会超过 9 x 9 = 81，不超过两位
4.每次只考虑两位，乘积 与 个位 相加
5.个位保留余数
6.十位保留取整，取整直接舍弃小数点，用0 效率，高于parseInt
7.最后while循环，删除多余的0

```
/**
 * @param {string} num1
 * @param {string} num2
 * @return {string}
 */
//  大整数
var multiply = function(num1, num2) {
    if(num1 == '0' || num2 == '0') return '0';
        let len1 = num1.length;
        let len2 = num2.length;
        let arr = new Array(len1 + len2).fill(0); //
        let  i =len1, j=len2;
        while(i){
            i--;
            while(j){
                j--;
                let sum = num1[i]*num2[j] + arr[i+j+1];
                arr[i+j] += 0 | sum/10;
                arr[i+j+1] = sum%10;
            }
```

```
            j=len2;
        }
        while(arr[0]==0){//while循环，删除多余的0
            arr.shift();
        }
        return arr.join('');
};
```

365. 水壶问题

1、只需要求出x和y的最大公约数d，并判断z是否是d的整数倍即可。

2、因此这道题可以完全转化为裴蜀定理。还是以题目给的例子x = 3, y = 5, z = 4，我们其实可以表示成 3 * 3 - 1 * 5 = 4, 即3 * x - 1 * y = z。我们用a和b分别表示3

升的水壶和5升的水壶。那么我们可以：

1、倒满a（1）

2、将a倒到b

3、再次倒满a（2）

4、再次将a倒到b（a这个时候还剩下1升）

5、倒空b（-1）

6、将剩下的1升倒到b

7、将a倒满（3）

8、将a倒到b

9、b此时正好是4升

10、上面的过程就是3 * x - 1 * y = z的具体过程解释。

```
var canMeasureWater = function(x, y, z) {
 if (x + y < z) return false;

 if (z === 0) return true;

 if (x === 0) return y === z;

 if (y === 0) return x === z;

 function GCD(a, b) {
   let min = Math.min(a, b);
   while (min) {
     if (a % min === 0 && b % min === 0) return min;
     min--;
   }
   return 1;
 }

 return z % GCD(x, y) === 0;
};
```

开课吧