【第四十五课】状态机模型与语言解释器(一)

1、2216. 美化数组的最少删除数

1. 一直保持，pre指针指向的是偶数下标的元素，last指针指向的是奇数下标的元素。
2. last指针指向len，存在两种情况：
3. pre指针指向倒数第二个元素，如果最后两个元素符合nums[i] != nums[i + 1],那么不需要再添加额外的操作数；反之，如果不符合，则会转换到pre指针指向倒数第一个元素的情况。
4. pre指针指向倒数第一个元素，pre指向的这个元素后面不存在元素，此时last已经指向了len，跳出循环，需要将pre指向的元素删除，还需要一个操作数。

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var minDeletion = function(nums) {
    let pre = 0,last = 1;
    let len = nums.length,count = 0;
    if(len == 1) return 1;
    while(last < len){
        // 指针为空 或者 指针的下一个没有
        if(nums[pre] != nums[last]){
            // 表示符合条件
            pre += 2;
            last += 2;
        }else{
            // 不符合
            count++;
            pre++;
            last++;
        }
    }
    if(pre == len - 1) count++;
    return count;
};
```

2、1562. 查找大小为 M 的最新分组

1. 我们可以用一个哈希表map来记录区间的两个端点，如果一个闭区间[a,b]中的元素全部为1，我们记录map(a) -> b且map(b) -> a，方便以后对区间进行合并
2. 用一个集合 `set` 来记录长度正好为 `m` 的区间端点；

3. 因为每次只会填充一个位置i，且每次填充的位置i均不相同，填充一个位置无非有四种情况：
   （1）i的左右两侧都为0（或超范围），也就是没有被记录的连续区间端点，那么填充的i独立成为一个长度为1的区间，记录map(i) -> i；
   （2）i左侧为0（或超范围），右侧与某一区间端点相邻；
   （3）i右侧为0（或超范围），左侧与某一区间端点相邻；
   （4）i的两端都与其他区间端点相邻，也就是i桥接了两个存在的区间。
4. 对于以上四种情况，我们可以对区间端点map进行更新。更新后的区间长度如果等于m，则将端点记录在set中；
5. map更新时，对当前影响到的区间端点，如果在set中存在，需要把他们删除，以为一个已经长度为m的区间，在更新后其长度一定大于m，不再符合题目要求；
6. 同时，如果set不为空，则说明存在长度为m的区间，记录set不为空的最后一个位置，返回即可。

```
/**
 * @param {number[]} arr
 * @param {number} m
 * @return {number}
 */
var findLatestStep = function(arr, m) {
    let map = new Map();
    let set = new Set();
    let res = -1;
    let n = arr.length;
    for(let i = 0;i < arr.length;i++){
        if((arr[i] === 1 || !map.has(arr[i] - 1)) && (arr[i] === n ||
!map.has(arr[i] + 1))){
            map.set(arr[i],arr[i]);
            if(m === 1) set.add(arr[i]);
        }else if(arr[i] === 1 || !map.has(arr[i] - 1)){
            set.delete(arr[i] + 1);
            set.delete(map.get(arr[i] + 1));
            map.set(arr[i],map.get(arr[i] + 1));
            map.set(map.get(arr[i] + 1),arr[i]);
            if(m === Math.abs(map.get(arr[i]) - arr[i]) + 1){
                set.add(arr[i]);
                set.add(map.get(arr[i]));
            }
        }else if(arr[i] === n || !map.has(arr[i] + 1)){
            set.delete(arr[i] - 1);
            set.delete(map.get(arr[i] - 1));
            map.set(arr[i],map.get(arr[i] - 1));
            map.set(map.get(arr[i] - 1),arr[i]);
           if(m === Math.abs(map.get(arr[i]) - arr[i]) + 1){
                set.add(arr[i]);
                set.add(map.get(arr[i]));
            }
        }else {
            let a = map.get(arr[i] - 1);
            let b = map.get(arr[i] + 1);
```

```
            set.delete(arr[i] + 1);
            set.delete(arr[i] - 1);
            set.delete(a);
            set.delete(b);
            map.delete(arr[i] - 1);
            map.delete(arr[i] + 1);
            map.set(a,b);
            map.set(b,a);
            if(m === Math.abs(a - b) + 1){
                set.add(a);
                set.add(b);
            }
        }
        if(set.size  > 0) res = i + 1;
    }
    return res;
};
```

3、1574. 删除最短的子数组使剩余数组有序

1. 用两个指针 i,j 指向左右区间的左边界，然后不断将 i 往右移动，缩小中间删除的区域。并记录下删除子数组元素的最小值。
2. 当 arr[i] > arr[j] 时，此时它们合并之后不再单调递增，这个时候右移 j 指针，始终保持最后合并的区间单调递增。

   直到某个指针到达右边界。

```
/**
 * @param {number[]} arr
 * @return {number}
 */
var findLengthOfShortestSubarray = function(arr) {
    let n = arr.length;
    let left = 0,right = n - 1;
    let res = n;

    while(left + 1 < n && arr[left] <= arr[left + 1]){
        left++;
    }
    if(left + 1 === n){
        return 0;
    }
    while(right > 0 && arr[right - 1] <= arr[right]){
        right--;
    }
```

```
    res = Math.min(right,n - left - 1);
    let i = 0,j = right;
    while(i <= left && j <= n - 1){
        if(arr[i] <= arr[j]){
            res = Math.min(res,j - i -1);
            i++;
        }else{
            j++;
        }
    }
    return res;
};
```

4、2226. 每个小孩最多能分到多少糖果

1. 由于可以拿走的最大糖果数目具有单向性，可以使用二分搜索。由于不限制分堆的次数，对于每个 candie 堆， 可以分出数量为 mid 个糖果堆 Math.floor(mid / n) 份，检查总数是否大于等于 k 就行了。

```
/**
 * @param {number[]} candies
 * @param {number} k
 * @return {number}
 */
var maximumCandies = function(candies, k) {
    const sum = candies.reduce((cur,next) => cur + next,0);
    if(sum < k){
        return 0;
    }
    let ans = 1;
    let lo = 1;
    let hi = Math.max(...candies);
    function check(n){
        let cnt = 0;
        for(let c of candies){
            cnt += Math.floor(c / n);
        }
        return cnt >= k;
    }
    while(lo <= hi){
        const mid = lo + ((hi - lo) >> 1);
        if(!check(mid)){
            hi = mid -1;
        }else{
            ans = mid;
            lo = mid + 1;
```

```
        }
    }
    return ans;
};
```