【第十七课】单调队列及经典问题

239. 滑动窗口最大值

1.当滑动窗口向右移动时，我们需要把一个新的元素放入队列中。为了保持队列的性质，我们会不断地将新的元素与队尾的元素相比较，如果前者大于等于后者，那么队尾的元素就可以被永久地移除，我们将其弹出队列。我们需要不断地进行此项操作，直到队列为空或者新的元素小于队尾的元素。

2.由于队列中下标对应的元素是严格单调递减的，因此此时队首下标对应的元素就是滑动窗口中的最大值。此时的最大值可能在滑动窗口左边界的左侧，并且随着窗口向右移动，它永远不可能出现在滑动窗口中了。因此我们还需要不断从队首弹出元素，直到队首元素在窗口中为止。

3.为了可以同时弹出队首和队尾的元素，这道题我们需要使用双端队列。满足这种单调性的双端队列一般称作「单调队列」。

```
var maxSlidingWindow = function(nums, k) {
    const n = nums.length;
    const q = [];
    for (let i = 0; i < k; i++) {
        while (q.length && nums[i] >= nums[q[q.length - 1]]) {
            q.pop();
        }
        q.push(i);
    }

    const ans = [nums[q[0]]];
    for (let i = k; i < n; i++) {
        while (q.length && nums[i] >= nums[q[q.length - 1]]) {
            q.pop();
        }
        q.push(i);
        while (q[0] <= i - k) {
            q.shift();
        }
        ans.push(nums[q[0]]);
    }
    return ans;
};
```

剑指 Offer 59 - II. 队列的最大值

1.从队列尾部插入元素时，我们可以提前取出队列中所有比这个元素小的元素，使得队列中只保留对结果有影响的数字。这样的方法等价于要求维持队列单调递减，即要保证每个元素的前面都没有比它小的元素。

2.我们只需要在插入每一个元素 value 时，从队列尾部依次取出比当前元素 value 小的元素，直到遇到一个比当前元素大的元素 value' 即可。

3.上面的过程需要从队列尾部取出元素，因此需要使用双端队列来实现。另外我们也需要一个辅助队列来记录所有被插入的值，以确定 pop_front 函数的返回值。

4.保证了队列单调递减后，求最大值时只需要直接取双端队列中的第一项即可。

```
 var MaxQueue = function() {
    this.queue1 = [];
    this.queue2 = [];
};

/**
 * @return {number}
 */
MaxQueue.prototype.max_value = function() {
    if (this.queue2.length) {
        return this.queue2[0];
    }
    return -1;
};

/**
 * @param {number} value
 * @return {void}
 */
MaxQueue.prototype.push_back = function(value) {
    this.queue1.push(value);
    while (this.queue2.length && this.queue2[this.queue2.length - 1] < value) {
        this.queue2.pop();
    }
    this.queue2.push(value);
};

/**
 * @return {number}
 */
MaxQueue.prototype.pop_front = function() {
    if (!this.queue1.length) {
        return -1;
    }
    const value = this.queue1.shift();
    if (value === this.queue2[0]) {
        this.queue2.shift();
    }
    return value;
};
```

```
/**
 * Your MaxQueue object will be instantiated and called as such:
 * var obj = new MaxQueue()
 * var param_1 = obj.max_value()
 * obj.push_back(value)
 * var param_3 = obj.pop_front()
 */
```

1438. 绝对差不超过限制的最长连续子数组

滑动窗口 + 单调队列

1、我们仅需要统计当前窗口内的最大值与最小值，因此我们也可以分别使用两个单调队列解决本题。

2、在实际代码中，我们使用一个单调递增的队列 queMin 维护最小值，一个单调递减的队列 queMax 维护最大值。这样我们只需要计算两个队列的队首的差值，即可知道当前窗口是否满足条件。

```
var longestSubarray = function(nums, limit) {
    const queMax = [];
    const queMin = [];
    const n = nums.length;
    let left = 0, right = 0;
    let ret = 0;
    while (right < n) {
        while (queMax.length && queMax[queMax.length - 1] < nums[right]) {
            queMax.pop();
        }
        while (queMin.length && queMin[queMin.length - 1] > nums[right]) {
            queMin.pop();
        }
        queMax.push(nums[right]);
        queMin.push(nums[right]);
        while (queMax.length && queMin.length && queMax[0] - queMin[0] > limit)
{
            if (nums[left] === queMin[0]) {
                queMin.shift();
            }
            if (nums[left] === queMax[0]) {
                queMax.shift();
            }
            left++;
        }
        ret = Math.max(ret, right - left + 1);
        right++;
    }
    return ret;
};
```

45. 跳跃游戏 II

1、当移动下标达到了当前覆盖的最远距离下标时，步数就要加一，来增加覆盖距离。最后的步数就是最少步数。

2、这里还是有个特殊情况需要考虑，当移动下标达到了当前覆盖的最远距离下标时

3、如果当前覆盖最远距离下标不是是集合终点，步数就加一，还需要继续走。

4、如果当前覆盖最远距离下标就是是集合终点，步数不用加一，因为不能再往后走了。

```
var jump = function(nums) {
    let curIndex = 0
    let nextIndex = 0
    let steps = 0
    for(let i = 0; i < nums.length - 1; i++) {
        nextIndex = Math.max(nums[i] + i, nextIndex)
        if(i === curIndex) {
            curIndex = nextIndex
            steps++
        }
    }

    return steps
};
```

93. 复原 IP 地址

由于我们需要找出所有可能复原出的 IP 地址，因此可以考虑使用回溯的方法，对所有可能的字符串分隔方式进行搜索，并筛选出满足要求的作为答案。

```
var restoreIpAddresses = function(s) {
    const SEG_COUNT = 4;
    const segments = new Array(SEG_COUNT);
    const ans = [];

    const dfs = (s, segId, segStart) => {
        // 如果找到了 4 段 IP 地址并且遍历完了字符串，那么就是一种答案
        if (segId === SEG_COUNT) {
            if (segStart === s.length) {
                ans.push(segments.join('.'));
            }
            return;
        }

        // 如果还没有找到 4 段 IP 地址就已经遍历完了字符串，那么提前回溯
        if (segStart === s.length) {
            return;
        }

        // 由于不能有前导零，如果当前数字为 0，那么这一段 IP 地址只能为 0
        if (s.charAt(segStart) === '0') {
            segments[segId] = 0;
            dfs(s, segId + 1, segStart + 1);
        }

        // 一般情况，枚举每一种可能性并递归
        let addr = 0;
        for (let segEnd = segStart; segEnd < s.length; ++segEnd) {
            addr = addr * 10 + (s.charAt(segEnd) - '0');
            if (addr > 0 && addr <= 0xFF) {
                segments[segId] = addr;
                dfs(s, segId + 1, segEnd + 1);
            } else {
                break;
```

```
            }
        }
    }

    dfs(s, 0, 0);
    return ans;
};
```