【第五课】堆与优先队列

正文视频从28分钟开始，2.27 -2.46 这个区间不用看，最后一道题因为我的马虎犯了一个错误，检查用了这么久，对不住我亲爱的同学们！！彩蛋题目和改正后的数据在文档的最后面。

面试题 17.20. 连续中值（https://leetcode-cn.com/problems/continuous-median-lcci/）

如果集合的个数是偶数个就是两个中间数相加除以2；如果是奇数个，中位数就是最中间的数。

```
//  大根堆+小根堆实现
 /**
 * initialize your data structure here.
 */
var MedianFinder = function () {
    // this.num = [];
    this.left = [-Infinity];
    this.right = [Infinity];
};

MedianFinder.prototype.resize = function () {
    if (this.left.length - this.right.length >= 2) {
        this.right.push(this.left.pop());
    } else if (this.right.length > this.left.length) {
        this.left.push(this.right.pop());
    }
}

/**
 * @param {number} num
 * @return {void}
 */
MedianFinder.prototype.addNum = function (num) {
    // this.num = [...this.num, num];
    if (num <= this.left[this.left.length - 1]) {
        this.left.push(num);
        this.left = this.left.sort((a, b) => a - b);
    } else {
        this.right.push(num);
        this.right = this.right.sort((a, b) => b - a);
    }
    this.resize();
};

/**
 * @return {number}
 */
MedianFinder.prototype.findMedian = function () {
    if (this.left.length === this.right.length) {
```

```
        return (this.left[this.left.length - 1] + this.right[this.right.length -
1]) / 2;
    } else  {
        return this.left[this.left.length - 1];
    }
};

/**
 * Your MedianFinder object will be instantiated and called as such:
 * var obj = new MedianFinder()
 * obj.addNum(num)
 * var param_2 = obj.findMedian()
 */
```

```
 /**
 * initialize your data structure here.
 */
var MedianFinder = function() {
    this.item = [];
};

/**
 * @param {number} num
 * @return {void}
 */
MedianFinder.prototype.addNum = function(num) {
     this.item.push(num);
};

/**
 * @return {number}
 */
MedianFinder.prototype.findMedian = function() {
    (this.item).sort(function(a,b){
        return a-b;
    });
    if((this.item.length % 2 == 0)){
        return ((this.item)[this.item.length /2] + (this.item)[this.item.length
/2 -1]) /2;
    }else{
        return (this.item)[(this.item.length -1)/2]
    }
};

/**
 * Your MedianFinder object will be instantiated and called as such:
 * var obj = new MedianFinder()
 * obj.addNum(num)
 * var param_2 = obj.findMedian()
 */
```

295. 数据流的中位数（https://leetcode-cn.com/problems/find-median-from-data-stream/）

做法和连续中值的一样

```
//  大根堆+小根堆实现
 /**
 * initialize your data structure here.
 */
var MedianFinder = function () {
    // this.num = [];
    this.left = [-Infinity];
    this.right = [Infinity];
};

MedianFinder.prototype.resize = function () {
    if (this.left.length - this.right.length >= 2) {
        this.right.push(this.left.pop());
    } else if (this.right.length > this.left.length) {
        this.left.push(this.right.pop());
    }
}

/**
 * @param {number} num
 * @return {void}
 */
MedianFinder.prototype.addNum = function (num) {
    // this.num = [...this.num, num];
    if (num <= this.left[this.left.length - 1]) {
        this.left.push(num);
        this.left = this.left.sort((a, b) => a - b);
    } else {
        this.right.push(num);
        this.right = this.right.sort((a, b) => b - a);
    }
    this.resize();
};

/**
 * @return {number}
 */
MedianFinder.prototype.findMedian = function () {
    if (this.left.length === this.right.length) {
        return (this.left[this.left.length - 1] + this.right[this.right.length -
1]) / 2;
    } else  {
        return this.left[this.left.length - 1];
    }
};

/**
 * Your MedianFinder object will be instantiated and called as such:
 * var obj = new MedianFinder()
```

```
 * obj.addNum(num)
 * var param_2 = obj.findMedian()
 */
```

```
/**
 * initialize your data structure here.
 */
var MedianFinder = function() {
    this.item = [];
};

/**
 * @param {number} num
 * @return {void}
 */
MedianFinder.prototype.addNum = function(num) {
     this.item.push(num);
};

/**
 * @return {number}
 */
MedianFinder.prototype.findMedian = function() {
    (this.item).sort(function(a,b){
        return a-b;
    });
    if((this.item.length % 2 == 0)){
        return ((this.item)[this.item.length /2] + (this.item)[this.item.length
/2 -1]) /2;
    }else{
        return (this.item)[(this.item.length -1)/2]
    }
};

/**
 * Your MedianFinder object will be instantiated and called as such:
 * var obj = new MedianFinder()
 * obj.addNum(num)
 * var param_2 = obj.findMedian()
 */
```

1801. 积压订单中的订单总数（https://leetcode-cn.com/problems/number-of-orders-in-the-backlog/）

主要考虑买货或和卖货：买进的话， 先看能卖的货 看有没有比要买的价格一样或低的， 可以抵消数量，抵消完了还有的话， 那就只能买进了；卖出的话， 先看之前买进的货 看有没有比要卖的价格一样或高的， 可以抵消， 抵消完了还有的话， 还是得卖；最后要求库存有多少件货。可以用最大优先队列存储要买进的货， 用最小优先队列存储要卖出的货， 方便拿出来抵消。优先队列直接使用lodash的priority-queue

```
/**
 * @param {number[][]} orders
 * @return {number}
 */
var getNumberOfBacklogOrders = function(orders) {
    let mod = 1000000007;
    let buy = new MaxPriorityQueue({priority:(bid) => bid.price});//采购
    let sell = new MinPriorityQueue({priority:(bid) => bid.price});//销售
    let total = 0;//最后的库存
    for(let [price, amount, orderType] of orders){
        if(orderType === 0){//采购
            while(!sell.isEmpty() && sell.front().priority <= price && amount
>0){
                let head = sell.dequeue().element;
                if(amount < head.amount){
                    sell.enqueue({price:head.price,amount:head.amount -
amount});
                    total -= amount;
                    amount = 0;
                }else{
                    amount -= head.amount;
                    total -= head.amount;
                }
            }
            if(amount > 0) buy.enqueue({price,amount}),total +=amount;
        }else{//orderType === 1 销售
           while(!buy.isEmpty() && buy.front().priority >= price && amount >0){
               let head = buy.dequeue().element;
               if(amount < head.amount){
                   buy.enqueue({price:head.price,amount:head.amount - amount});
                   total -=amount;
                   amount = 0;
               }else{
                    amount -= head.amount;
                    total -= head.amount;
               }
           }
           if(amount > 0) sell.enqueue({price,amount}),total += amount;
        }
    }

    return total  % mod;
};
```

264. 丑数 II（https://leetcode-cn.com/problems/ugly-number-ii/）

跟第二课里面第k个数的题一样,素因子这个题，只不过里面的素因子的的值变化了。首先声明几个变量，是用来记录每个素因子的使用次数，接着遍历N次，每次进行素因子相乘，取最小值。最后进行去重，如果有重复的数，就在当前素因子，使用次数上+1。

```
/**
 * @param {number} n
```

```
 * @return {number}
 */
// 注意这个里面的去重；和我们第二课的面试题 17.09. 第 k 个数,一样
var nthUglyNumber = function(n) {
    var dp = new Array();
    dp[0] = 1;
    var p2 = 0;
    var p3 = 0;
    var p5 = 0;
    for(let i = 1; i < n;i++){
        dp[i] = Math.min(dp[p2]*2,Math.min(dp[p3]*3,dp[p5]*5));
        // 去重
        if(dp[i] === dp[p2]*2) p2++;
        if(dp[i] === dp[p3]*3) p3++;
        if(dp[i] === dp[p5]*5) p5++;
    }
    return dp[n-1];
};
```

313. 超级丑数（https://leetcode-cn.com/problems/super-ugly-number/）

这道题和丑数2的思路和一样。就是素因子是一个数组，也是得注意去重。这里面注意Math.min.apply(null,arr) 可以求最小值，使用apply的优点是在部分JS引擎中提升性能。

```
/**
 * @param {number} n
 * @param {number[]} primes
 * @return {number}
 */
// 也是新建数组，存储我们素因子想乘或者自乘的值，去重；
var nthSuperUglyNumber = function(n, primes) {
    const res = [1];
    const points  = new Array(primes.length).fill(0);//创建一个数组，赋值
    let min,map;
    for(let i = 1;i < n;i++){
        map = primes.map((prime,index) => res[points[index]]* prime )
        min = Math.min.apply(null,map);//Math.min.apply，求最小值
        // 去重
        primes.forEach((prime,index)=>{
            if(map[index] === min) points[index]++;
        })
        res.push(min);
    }
    return res[n-1];
};
```

1753. 移除石子的最大得分（https://leetcode-cn.com/problems/maximum-score-from-removing-stones/）

首先对三个值进行排序，按着从小到大的排序方便我们以后取值。首先干掉第一堆里面，第三根堆比第二堆长的数量。接着判断第一堆里面是否为0，否就是第二堆和第三堆的数量是一样的，分别消掉第一堆里面的二分之一个部分，此时第一堆被削掉了，然后不断地减去第二堆和第三堆的数量。最后返回轮数。

```
/**
 * @param {number} a
 * @param {number} b
 * @param {number} c
 * @return {number}
 */
var maximumScore = function(a, b, c) {
    if(a>b) [a,b] = [b,a];
    if(a>c) [a,c] = [c,a];
    if(b>c) [b,c] = [c,b];
    var ans = 0;
    // step1
    var cnt1 = Math.min(c-b,a);
    a -= cnt1;
    c -= cnt1;
    ans += cnt1;
    // step 2
    if(a !=0){
        if( a % 2 ==1) a-=1;
        b -= a/2 |0;
        c -= a/2 |0;
        ans += a;
    }
    // step3
    ans += b;
    return ans;
};
```

题目描述：给出一个二叉树的中序序列和后序序列，请推导出二叉树的前序序列，并将前序序列存储到数组中，求出数组元素和下标相乘的累加之和。（数组索引从0开始）

```
 postOrder traversal 103857 579220 634827 591899 787863 1046307 1591804 1456881
1382996 1279694 1158931 827654 1654115 1858301 2309657 2343977 2317931 2267835
1767859 2534345 2532162 2437415 2424201 1612718 2770175 2833362 3014150 3393602
3126711 3482580 2881637 2871682 2826584 3733657 3523647 3834018 2764278 3920601
4208987 4309938 4297111 4225518 3866869 4496929 4522001 4435967 5066986 4986504
4746509 5075516 4547481 5332515 5474284 5255939 5655006 5811063 5949618 5702782
6028856 5541875 5233892 6196531 6213872 6350155 6477928 6418966 6626558 6532974
6983104 6280298 7246794 7530932 7653296 7299049 7295593 7732415 7748116 8208637
7691842 7120444 7086416 6040500 4407246 8409970 8414634 8318376 8989734 9076914
9463496 9342589 9608664 9253016 9232618 9193845 9772869 9861978 8579250 8220339
3854965 2678011

inOrder traversal 103857 579220 591899 634827 787863 827654 1046307 1158931
1279694 1382996 1456881 1591804 1612718 1654115 1767859 1858301 2267835 2309657
2317931 2343977 2424201 2437415 2532162 2534345 2678011 2764278 2770175 2826584
2833362 2871682 2881637 3014150 3126711 3393602 3482580 3523647 3733657 3834018
3854965 3866869 3920601 4208987 4225518 4297111 4309938 4407246 4435967 4496929
4522001 4547481 4746509 4986504 5066986 5075516 5233892 5255939 5332515 5474284
5541875 5655006 5702782 5811063 5949618 6028856 6040500 6196531 6213872 6280298
6350155 6418966 6477928 6532974 6626558 6983104 7086416 7120444 7246794 7295593
7299049 7530932 7653296 7691842 7732415 7748116 8208637 8220339 8318376 8409970
8414634 8579250 8989734 9076914 9193845 9232618 9253016 9342589 9463496 9608664
9772869 9861978
```