1825. Finding MK Average

My Submissions (/contest/weekly-contest-236/problems/finding-mk-average/submissions/) Back to Contest (/contest/weekly-contest-236/)

User Accepted:	158
User Tried:	1288
Total Accepted:	174
Total Submissions:	3173
Difficulty:	Hard

You are given two integers, `m` and `k`, and a stream of integers. You are tasked to implement a data structure that calculates the **MKAverage** for the stream.

The **MKAverage** can be calculated using these steps:

1. If the number of the elements in the stream is less than `m` you should consider the **MKAverage** to be `-1`. Otherwise, copy the last `m` elements of the stream to a separate container.
2. Remove the smallest `k` elements and the largest `k` elements from the container.
3. Calculate the average value for the rest of the elements **rounded down to the nearest integer**.

Implement the `MKAverage` class:

- `MKAverage(int m, int k)` Initializes the **MKAverage** object with an empty stream and the two integers `m` and `k`.
- `void addElement(int num)` Inserts a new element `num` into the stream.
- `int calculateMKAverage()` Calculates and returns the **MKAverage** for the current stream **rounded down to the nearest integer**.

Example 1:

```
Input
["MKAverage", "addElement", "addElement", "calculateMKAverage", "addElement", "calculateMKAverage", "addElement", "addElement",
[[3, 1], [3], [1], [], [10], [], [5], [5], [5], []]
Output
[null, null, null, -1, null, 3, null, null, null, 5]

Explanation
MKAverage obj = new MKAverage(3, 1);
obj.addElement(3);        // current elements are [3]
obj.addElement(1);        // current elements are [3,1]
obj.calculateMKAverage(); // return -1, because m = 3 and only 2 elements exist.
obj.addElement(10);       // current elements are [3,1,10]
obj.calculateMKAverage(); // The last 3 elements are [3,1,10].
                          // After removing smallest and largest 1 element the container will be [3].
                          // The average of [3] equals 3/1 = 3, return 3
obj.addElement(5);        // current elements are [3,1,10,5]
obj.addElement(5);        // current elements are [3,1,10,5,5]
obj.addElement(5);        // current elements are [3,1,10,5,5,5]
obj.calculateMKAverage(); // The last 3 elements are [5,5,5].
                          // After removing smallest and largest 1 element the container will be [5].
                          // The average of [5] equals 5/1 = 5, return 5
```

Constraints:

- $3 <= m <= 10^5$
- $1 <= k*2 < m$
- $1 <= num <= 10^5$
- At most 10^5 calls will be made to `addElement` and `calculateMKAverage`.

Discuss (https://leetcode.com/problems/finding-mk-average/discuss)

JavaScript

```
function Bisect() {
    return { insort_right, insort_left, bisect_left, bisect_right }
    function insort_right(a, x, lo = 0, hi = null) {
        lo = bisect_right(a, x, lo, hi);
        a.splice(lo, 0, x);
    }
    function bisect_right(a, x, lo = 0, hi = null) { // > upper_bound
        if (lo < 0) throw new Error('lo must be non-negative');
```

```
        if (hi == null) hi = a.length;
        while (lo < hi) {
            let mid = parseInt((lo + hi) / 2);
            a[mid] > x ? hi = mid : lo = mid + 1;
        }
        return lo;
    }
    function insort_left(a, x, lo = 0, hi = null) {
        lo = bisect_left(a, x, lo, hi);
        a.splice(lo, 0, x);
    }
    function bisect_left(a, x, lo = 0, hi = null) { // >= lower_bound
        if (lo < 0) throw new Error('lo must be non-negative');
        if (hi == null) hi = a.length;
        while (lo < hi) {
            let mid = parseInt((lo + hi) / 2);
            a[mid] < x ? lo = mid + 1 : hi = mid;
        }
        return lo;
    }
}

function MultiSet(elements) {
    let a = [], m = new Map(), bi = new Bisect();
    initialize();
    return { insert, first, last, get, search, poll, pollLast, lower_bound, upper_bound, findKth, eraseByIndex,
eraseAll, contains, size, clear, show };
    function initialize() {
        if (elements) {
            for (const x of elements) {
                bi.insort_right(a, x);
                m.set(x, m.get(x) + 1 || 1);
            }
        }
    }
    function insert(x) {
        bi.insort_right(a, x);
        m.set(x, m.get(x) + 1 || 1);
    }
    function first() {
        return a[0];
    }
    function last() {
        return a[a.length - 1];
    }
    function get(i) {
        return a[i];
    }
    function poll() {
        let res = a[0];
        a.splice(0, 1);
        removeOneOrManyMap(m, res);
        return res;
    }
    function pollLast() {
        let res = a.pop();
        removeOneOrManyMap(m, res);
        return res;
    }
    function lower_bound(x) {
        return bi.bisect_left(a, x);
    }
    function upper_bound(x) {
        return bi.bisect_right(a, x);
    }
    function findKth(k) {
        return a[k - 1];
    }
    function search(x) {
        let idx = lower_bound(x);
        return idx == a.length ? idx - 1 : idx;
    }
    function eraseByIndex(idx) {
        removeOneOrManyMap(m, a[idx]);
        a.splice(idx, 1);
    }
    function eraseAll(x) {
```

```
        if (contains(x)) {
            let idx = search(x), occ = m.get(x);
            while (occ--) a.splice(idx, 1);
            m.delete(x);
        }
    }
    function removeOneOrManyMap(m, x, cnt = 1) {
        let occ = m.get(x);
        occ > cnt ? m.set(x, occ - cnt) : m.delete(x);
    }
    function contains(x) {
        return m.has(x);
    }
    function size() {
        return a.length;
    }
    function clear() {
        a = [];
        m.clear();
    }
    function show() {
        return a;
    }
}

function MKAverage(m, k) {
    let L = new MultiSet(), R = new MultiSet(), M = new MultiSet(), a = [], sum = 0, pos = 0, sz = m - 2 * k;
    return { addElement, calculateMKAverage }
    function addElement(x) {
        add(x);
        if (pos >= m) remove(a[pos % m]);
        a[pos++ % m] = x;
    }
    function calculateMKAverage() {
        if (pos < m) return -1;
        return sum / sz >> 0;
    }
    function add(x) {
        L.insert(x);
        if (L.size() > k) {
            let idx = L.size() - 1, v = L.get(idx);
            M.insert(v);
            sum += v;
            L.eraseByIndex(idx);
        }
        if (M.size() > sz) {
            let idx = M.size() - 1, v = M.get(idx);
            sum -= v;
            R.insert(v);
            M.eraseByIndex(idx);
        }
    }
    function remove(x) {
        if (x <= L.last()) {
            L.eraseByIndex(L.search(x));
        } else if (x <= M.last()) {
            let idx = M.search(x), v = M.get(idx);
            sum -= v;
            M.eraseByIndex(idx);
        } else {
            R.eraseByIndex(R.search(x));
        }
        if (L.size() < k) {
            let v = M.first();
            L.insert(v);
            sum -= v;
            M.poll();
        }
        if (M.size() < sz) {
            let v = R.first();
            M.insert(v);
            sum += v;
            R.poll();
        }
    }
}
```

Custom Testcase Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/877906698/) More Details (/submissions/detail/877906698/)

Share your acceptance!