6066. Count Integers in Intervals

My Submissions (/contest/weekly-contest-293/problems/count-integers-in-intervals/submissions/) | Back to Contest (/contest/weekly-contest-293/)

User Accepted:	0
User Tried:	0
Total Accepted:	0
Total Submissions:	0
Difficulty:	Hard

Given an **empty** set of intervals, implement a data structure that can:

- **Add** an interval to the set of intervals.
- **Count** the number of integers that are present in **at least one** interval.

Implement the `CountIntervals` class:

- `CountIntervals()` Initializes the object with an empty set of intervals.
- `void add(int left, int right)` Adds the interval `[left, right]` to the set of intervals.
- `int count()` Returns the number of integers that are present in **at least one** interval.

Note that an interval `[left, right]` denotes all the integers `x` where `left <= x <= right` .

Example 1:

```
Input
["CountIntervals", "add", "add", "count", "add", "count"]
[[], [2, 3], [7, 10], [], [5, 8], []]
Output
[null, null, null, 6, null, 8]

Explanation
CountIntervals countIntervals = new CountIntervals(); // initialize the object with an empty set of intervals.
countIntervals.add(2, 3);  // add [2, 3] to the set of intervals.
countIntervals.add(7, 10); // add [7, 10] to the set of intervals.
countIntervals.count();    // return 6
                           // the integers 2 and 3 are present in the interval [2, 3].
                           // the integers 7, 8, 9, and 10 are present in the interval [7, 10].
countIntervals.add(5, 8);  // add [5, 8] to the set of intervals.
countIntervals.count();    // return 8
                           // the integers 2 and 3 are present in the interval [2, 3].
                           // the integers 5 and 6 are present in the interval [5, 8].
                           // the integers 7 and 8 are present in the intervals [5, 8] and [7, 10].
                           // the integers 9 and 10 are present in the interval [7, 10].
```

Constraints:

- `1 <= left <= right <= 10`^{9}
- At most `10`^{5} calls **in total** will be made to `add` and `count` .
- At least **one** call will be made to `count` .

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 TreeMap(elements) {
    let ts = [], m = new Map(), bisect = new Bisect();
    build();
    return { put, ceilingKey, higherKey, lowerKey, floorKey, ceilingEntry, higherEntry, lowerEntry, floorEntry, remove, contains, size, clear, show };
    function build() {
        if (elements) {
            for (const e of elements) {
                if (!m.has(e)) bisect.insort_right(ts, e);
                addOneOrManyMap(m, e);
            }
        }
    }
    function put(k, v) {
        bisect.insort_right(ts, k);
        m.set(k, v);
    }
    function ceilingKey(e) { // >= lower_bound
        let idx = bisect.bisect_right(ts, e);
        let res = ts[idx - 1] == e ? e : ts[bisect.bisect_right(ts, e)];
        return res == undefined ? null : res;
    }
    function higherKey(e) { // > upper_bound
        let idx = bisect.bisect_right(ts, e);
        let res = ts[idx] > e ? ts[idx] : ts[bisect.bisect_right(ts, e) + 1];
        return res == undefined ? null : res;
    }
    function floorKey(e) { // <=
        let idx = bisect.bisect_left(ts, e);
        let res = ts[idx] == e ? e : ts[bisect.bisect_left(ts, e) - 1];
        return res == undefined ? null : res;
    }
    function lowerKey(e) { // <
        let idx = bisect.bisect_left(ts, e);
        let res = ts[idx] < e ? ts[idx] : ts[bisect.bisect_left(ts, e) - 1];
        return res == undefined ? null : res;
    }
    function data(k) {
        return k == null ? null : { key: k, value: m.get(k) }
    }
    function ceilingEntry(k) {
        return data(ceilingKey(k));
    }
    function higherEntry(k) {
        return data(higherKey(k));
    }
    function floorEntry(k) {
        return data(floorKey(k));
    }
    function lowerEntry(k) {
        return data(lowerKey(k));
    }
    function remove(e) {
        let idx = bisect.bisect_left(ts, e);
        if (ts[idx] == e) ts.splice(idx, 1);
        removeOneOrManyMap(m, e);
    }
    function contains(e) {
        return m.has(e);
    }
    function size() {
        return ts.length;
    }
    function clear() {
        ts = [];
        m.clear();
    }
    function show() {
        let res = new Map();
        for (const x of ts) res.set(x, m.get(x));
        return res;
    }
    function addOneOrManyMap(m, x, cnt = 1) {
        return m.set(x, m.get(x) + cnt || cnt);
    }
    function removeOneOrManyMap(m, x, cnt = 1) {
        let occ = m.get(x);
        occ > cnt ? m.set(x, occ - cnt) : m.delete(x);
    }
}


function CountIntervals() {
    let tm = new TreeMap(), cnt = 0;
    return { add, count }
    function add(left, right) {
        let lower = tm.floorEntry(left);
        if (lower != null && lower.value >= left) {
            let k = lower.key, v = lower.value;
            cnt -= v - k + 1;
            left = Math.min(left, k);
            right = Math.max(right, v);
            tm.remove(k);
        }
        while (1) {
           let higher = tm.ceilingEntry(left);
           if (higher == null || higher.key > right) break;
           let k = higher.key, v = higher.value;
           tm.remove(k);
           cnt -= v - k + 1;
           right = Math.max(right, v);
        }
        cnt += right - left + 1;
        tm.put(left, right);
    }
    function count() {
        return cnt;
    }
}
```

□ **Custom Testcase** Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/699856399/) More Details (/submissions/detail/699856399/)

Share your acceptance!

3

Help Center (/support) | Jobs (/jobs) | Bug Bounty (/bugbounty) | Online Interview (/interview/) | Students (/student) | Terms (/terms) | Privacy Policy (/privacy)

United States (/region)