6113. Smallest Number in Infinite Set

My Submissions (/contest/weekly-contest-301/problems/smallest-number-in-infinite-set/submissions/) Back to Contest (/contest/weekly-contest-301/)

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

You have a set which contains all positive integers `[1, 2, 3, 4, 5, ...]` .

Implement the `SmallestInfiniteSet` class:

- `SmallestInfiniteSet()` Initializes the **SmallestInfiniteSet** object to contain **all** positive integers.
- `int popSmallest()` **Removes** and returns the smallest integer contained in the infinite set.
- `void addBack(int num)` **Adds** a positive integer `num` back into the infinite set, if it is **not** already in the infinite set.

Example 1:

```
Input
["SmallestInfiniteSet", "addBack", "popSmallest", "popSmallest", "popSmallest", "addBack", "popSmallest", "popSmallest", "popSm
[[], [2], [], [], [], [1], [], [], []]
Output
[null, null, 1, 2, 3, null, 1, 4, 5]

Explanation
SmallestInfiniteSet smallestInfiniteSet = new SmallestInfiniteSet();
smallestInfiniteSet.addBack(2);    // 2 is already in the set, so no change is made.
smallestInfiniteSet.popSmallest(); // return 1, since 1 is the smallest number, and remove it from the set.
smallestInfiniteSet.popSmallest(); // return 2, and remove it from the set.
smallestInfiniteSet.popSmallest(); // return 3, and remove it from the set.
smallestInfiniteSet.addBack(1);    // 1 is added back to the set.
smallestInfiniteSet.popSmallest(); // return 1, since 1 was added back to the set and
                                   // is the smallest number, and remove it from the set.
smallestInfiniteSet.popSmallest(); // return 4, and remove it from the set.
smallestInfiniteSet.popSmallest(); // return 5, and remove it from the set.
```

Constraints:

- `1 <= num <= 1000`
- At most `1000` calls will be made **in total** to `popSmallest` and `addBack` .

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 TreeSet(elements) {
    let ts = [], se = new Set(), bisect = new Bisect();
    initialize();
    return { add, first, last, poll, pollLast, floor, ceiling, lower, higher, remove, contains, size, clear, show };
    function initialize() {
        if (elements) {
            for (const e of elements) {
                if (!se.has(e)) {
                    bisect.insort_right(ts, e);
                    se.add(e);
                }
            }
        }
    }
    function add(e) {
        if (!se.has(e)) {
            bisect.insort_right(ts, e);
            se.add(e);
        }
    }
    function first() {
        return ts[0];
    }
    function last() {
        return ts[ts.length - 1];
    }
    function poll() {
        let res = ts[0];
        ts.splice(0, 1);
        se.delete(res);
        return res;
    }
    function pollLast() {
        let res = ts.pop();
        se.delete(res);
        return res;
    }
    function ceiling(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 higher(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 floor(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 lower(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 remove(e) {
        let idx = bisect.bisect_left(ts, e);
        if (ts[idx] == e) ts.splice(idx, 1);
        se.delete(e);
    }
    function contains(e) {
        return se.has(e);
    }
    function size() {
        return ts.length;
    }
    function clear() {
        ts = [];
        se.clear();
    }
    function show() {
        return ts;
    }
}

function SmallestInfiniteSet() {
    let ts = new TreeSet();
    for (let i = 1; i <= 1000; i++) ts.add(i);
    return { popSmallest, addBack }
    function popSmallest() {
        return ts.poll();
    }
    function addBack(x) {
        ts.add(x);
    }
}
```

Custom Testcase Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/743038744/) More Details (/submissions/detail/743038744/)

Share your acceptance!