2571. Minimum Operations to Reduce an Integer to 0

My Submissions (/contest/weekly-contest-333/problems/minimum-operations-to-reduce-an-integer-to-0/submissions/)

Back to Contest (/contest/weekly-contest-333/)

User Accepted:	5071
User Tried:	7383
Total Accepted:	5313
Total Submissions:	14651
Difficulty:	Easy

You are given a positive integer `n`, you can do the following operation **any** number of times:

- Add or subtract a **power** of `2` from `n`.

Return *the* ***minimum*** *number of operations to make* `n` *equal to* `0`.

A number `x` is power of `2` if `x == 2^i` where `i >= 0`.

Example 1:

```
Input: n = 39
Output: 3
Explanation: We can do the following operations:
- Add 2^0 = 1 to n, so now n = 40.
- Subtract 2^3 = 8 from n, so now n = 32.
- Subtract 2^5 = 32 from n, so now n = 0.
It can be shown that 3 is the minimum number of operations we need to make n equal to 0.
```

Example 2:

```
Input: n = 54
Output: 3
Explanation: We can do the following operations:
- Add 2^1 = 2 to n, so now n = 56.
- Add 2^3 = 8 to n, so now n = 64.
- Subtract 2^6 = 64 from n, so now n = 0.
So the minimum number of operations is 3.
```

Constraints:

- `1 <= n <= 10^5`

Discuss (https://leetcode.com/problems/minimum-operations-to-reduce-an-integer-to-0/discuss)

JavaScript

```
const cutMaxConsecutive = (as) => { let d = [], l = 0, n = as.length; for (let i = 0; i + 1 < n; i++) { if (as[i + 1] != as[i]) { d.push(as.slice(l, i + 1)); l = i + 1; } } d.push(as.slice(l)); return d; };

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, has, 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 has(e) {
        return se.has(e);
    }
    function size() {
```

```
        return ts.length;
    }
    function clear() {
        ts = [];
        se.clear();
    }
    function show() {
        return ts;
    }
}

const minOperations = (N) => {
    let s = N.toString(2), n = s.length, ts = new TreeSet(), cnt = 0, d = cutMaxConsecutive(s), res1 = 0;
    for (const e of d) {
        if (e[0] == '1') {
            res1 += e.length == 1 ? 1 : 2;
        }
    }
    for (let i = 0; i < n; i++) {
        if (s[i] == '1') ts.add(i);
    }
    // pr(s, ts.show());
    // for (let t = 1; t <= 10; t++) {
    while (ts.size() > 1) {
        // pr(ts.show(), "cnt", cnt);
        while (1) {
            let cur = ts.last(), pre = ts.lower(cur);
            // pr("cur", cur, "pre", pre, ts.show());
            ts.remove(cur);
            if (cur - 1 == pre) {
            } else { // 7 4
                let last = cur - 1;
                if (last - 1 == pre) {
                    ts.add(last);
                } else {
                    cnt++; // single
                }
                break;
            }
        }
        cnt++; // merge
    }
    let res2 = cnt + ts.size();
    // pr("222", ts.show(), "cnt", cnt, "res2", res2, "res1", res1);
    return Math.min(res1, res2);
};
```

Custom Testcase Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/900812220/) More Details > (/submissions/detail/900812220/)

Share your acceptance!