2410. Maximum Matching of Players With Trainers

My Submissions (/contest/biweekly-contest-87/problems/maximum-matching-of-players-with-trainers/submissions/)

Back to Contest (/contest/biweekly-contest-87/)

You are given a **0-indexed** integer array `players`, where `players[i]` represents the **ability** of the `i`th player. You are also given a **0-indexed** integer array `trainers`, where `trainers[j]` represents the **training capacity** of the `j`th trainer.

The `i`th player can **match** with the `j`th trainer if the player's ability is **less than or equal to** the trainer's training capacity. Additionally, the `i`th player can be matched with at most one trainer, and the `j`th trainer can be matched with at most one player.

Return *the **maximum** number of matchings between* `players` *and* `trainers` *that satisfy these conditions.*

User Accepted:	9867
User Tried:	11124
Total Accepted:	10182
Total Submissions:	18846
Difficulty:	Medium

Example 1:

```
Input: players = [4,7,9], trainers = [8,2,5,8]
Output: 2
Explanation:
One of the ways we can form two matchings is as follows:
- players[0] can be matched with trainers[0] since 4 <= 8.
- players[1] can be matched with trainers[3] since 7 <= 8.
It can be proven that 2 is the maximum number of matchings that can be formed.
```

Example 2:

```
Input: players = [1,1,1], trainers = [10]
Output: 1
Explanation:
The trainer can be matched with any of the 3 players.
Each player can only be matched with one trainer, so the maximum answer is 1.
```

Constraints:

- `1 <= players.length, trainers.length <=` 10^5
- `1 <= players[i], trainers[j] <=` 10^9

Discuss (https://leetcode.com/problems/maximum-matching-of-players-with-trainers/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 TreeMap(g) {
    let ts = [], m = new Map(), bisect = new Bisect();
    initialize();
    return { set, get, firstKey, lastKey, keys, pollFirstEntry, pollLastEntry, ceilingKey, higherKey, lowerKey, floorKey,
ceilingEntry, higherEntry, lowerEntry, floorEntry, remove, contains, size, clear, show };
    function initialize() {
        if (g) {
            for (const [k, v] of g) {
                if (!m.has(k)) bisect.insort_right(ts, k);
                m.set(k, v);
            }
        }
    }
    function set(k, v) {
        if (!m.has(k)) bisect.insort_right(ts, k); // ts has no duplicates/unique key
        m.set(k, v); // update key with most recent value
    }
    function get(k) {
        return m.get(k);
    }
    function keys() {
        return ts;
    }
    function firstKey() {
        return ts[0];
    }
    function lastKey() {
        return ts[ts.length - 1];
    }
    function pollFirstEntry() {
        let k = ts[0], v = m.get(k);
        ts.splice(0, 1);
        m.delete(k);
        return [k, v];
    }
    function pollLastEntry() {
        let k = ts.pop(), v = m.get(k);
        m.delete(k);
        return [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);
        m.delete(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;
    }
}

const addOneOrManyMap = (m, x, cnt = 1) => m.set(x, m.get(x) + cnt || cnt);
const removeOneOrManyMap = (m, x, cnt = 1) => { let occ = m.get(x); occ > cnt ? m.set(x, occ - cnt) : m.remove(x); };

const matchPlayersAndTrainers = (a, b) => {
   let m = new TreeMap(), res = 0;
   for (const x of b) addOneOrManyMap(m, x);
   for (const x of a) {
       let next = m.ceilingKey(x);
       if (next != null) {
           res++;
           removeOneOrManyMap(m, next);
       }
   }
   return res;
};
```

Custom Testcase Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/802360533/) More Details (/submissions/detail/802360533/)

Share your acceptance!

2