5937. Sequentially Ordinal Rank Tracker

My Submissions (/contest/biweekly-contest-67/problems/sequentially-ordinal-rank-tracker/submissions/)

Back to Contest (/contest/biweekly-contest-67/)

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

A scenic location is represented by its `name` and attractiveness `score`, where `name` is a **unique** string among all locations and `score` is an integer. Locations can be ranked from the best to the worst. The **higher** the score, the better the location. If the scores of two locations are equal, then the location with the **lexicographically smaller** name is better.

You are building a system that tracks the ranking of locations with the system initially starting with no locations. It supports:

- **Adding** scenic locations, **one at a time**.
- **Querying** the i^{th} **best** location of **all locations already added**, where `i` is the number of times the system has been queried (including the current query).
  - For example, when the system is queried for the 4^{th} time, it returns the 4^{th} best location of all locations already added.

Note that the test data are generated so that **at any time**, the number of queries **does not exceed** the number of locations added to the system.

Implement the `SORTracker` class:

- `SORTracker()` Initializes the tracker system.
- `void add(string name, int score)` Adds a scenic location with `name` and `score` to the system.
- `string get()` Queries and returns the i^{th} best location, where `i` is the number of times this method has been invoked (including this invocation).

Example 1:

```
Input
["SORTracker", "add", "add", "get", "add", "get", "add", "get", "add", "get", "add", "get", "get"]
[[], ["bradford", 2], ["branford", 3], [], ["alps", 2], [], ["orland", 2], [], ["orlando", 3], [], ["alpine", 2], [],
Output
[null, null, null, "branford", null, "alps", null, "bradford", null, "bradford", null, "bradford", "orland"]

Explanation
SORTracker tracker = new SORTracker(); // Initialize the tracker system.
tracker.add("bradford", 2); // Add location with name="bradford" and score=2 to the system.
tracker.add("branford", 3); // Add location with name="branford" and score=3 to the system.
tracker.get();              // The sorted locations, from best to worst, are: branford, bradford.
                            // Note that branford precedes bradford due to its higher score (3 > 2).
                            // This is the 1st time get() is called, so return the best location: "branford".
tracker.add("alps", 2);     // Add location with name="alps" and score=2 to the system.
tracker.get();              // Sorted locations: branford, alps, bradford.
                            // Note that alps precedes bradford even though they have the same score (2).
                            // This is because "alps" is lexicographically smaller than "bradford".
                            // Return the 2nd best location "alps", as it is the 2nd time get() is called.
tracker.add("orland", 2);   // Add location with name="orland" and score=2 to the system.
tracker.get();              // Sorted locations: branford, alps, bradford, orland.
                            // Return "bradford", as it is the 3rd time get() is called.
tracker.add("orlando", 3);  // Add location with name="orlando" and score=3 to the system.
tracker.get();              // Sorted locations: branford, orlando, alps, bradford, orland.
                            // Return "bradford".
tracker.add("alpine", 2);   // Add location with name="alpine" and score=2 to the system.
tracker.get();              // Sorted locations: branford, orlando, alpine, alps, bradford, orland.
                            // Return "bradford".
tracker.get();              // Sorted locations: branford, orlando, alpine, alps, bradford, orland.
                            // Return "orland".
```

Constraints:

- `name` consists of lowercase English letters, and is unique among all locations.
- `1 <= name.length <= 10`
- `1 <= score <= 10^5`
- At any time, the number of calls to `get` does not exceed the number of calls to `add`.
- At most `4 * 10^4` calls **in total** will be made to `add` and `get`.

JavaScript

```
const comp = (x, y) => {
    if (x[1] != y[1]) return y[1] - x[1];
    return x[0].localeCompare(y[0]);
};

const comp2 = (x, y) => {
    if (x[1] != y[1]) return x[1] - y[1];
    return y[0].localeCompare(x[0]);
};

function SORTracker() {
    let pq = new MinPriorityQueue({ compare: comp });
    let best = new MinPriorityQueue({ compare: comp2 });
    return { add, get }
    function add(name, score) {
        pq.enqueue([name, score]);
        while (best.size() && comp(pq.front(), best.front()) < 0) {
            let b = best.dequeue(), a = pq.dequeue();
            best.enqueue(a);
            pq.enqueue(b);
        }
    }
    function get() {
        let res = pq.dequeue();
        best.enqueue(res);
        return res[0];
    }
}
```

Custom Testcase Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/600360868/) More Details (/submissions/detail/600360868/)

Share your acceptance!