5783. Design Movie Rental System

My Submissions (/contest/biweekly-contest-55/problems/design-movie-rental-system/submissions/)

Back to Contest (/contest/biweekly-contest-55/)

User Accepted:	0
User Tried:	1
Total Accepted:	0
Total Submissions:	1
Difficulty:	Hard

You have a movie renting company consisting of `n` shops. You want to implement a renting system that supports searching for, booking, and returning movies. The system should also support generating a report of the currently rented movies.

Each movie is given as a 2D integer array `entries` where `entries[i] = [shop_i, movie_i, price_i]` indicates that there is a copy of movie `movie_i` at shop `shop_i` with a rental price of `price_i`. Each shop carries **at most one** copy of a movie `movie_i`.

The system should support the following functions:

- **Search**: Finds the **cheapest 5 shops** that have an **unrented copy** of a given movie. The shops should be sorted by **price** in ascending order, and in case of a tie, the one with the **smaller** `shop_i` should appear first. If there are less than 5 matching shops, then all of them should be returned. If no shop has an unrented copy, then an empty list should be returned.
- **Rent**: Rents an **unrented copy** of a given movie from a given shop.
- **Drop**: Drops off a **previously rented copy** of a given movie at a given shop.
- **Report**: Returns the **cheapest 5 rented movies** (possibly of the same movie ID) as a 2D list `res` where `res[j] = [shop_j, movie_j]` describes that the j^{th} cheapest rented movie `movie_j` was rented from the shop `shop_j`. The movies in `res` should be sorted by **price** in ascending order, and in case of a tie, the one with the **smaller** `shop_j` should appear first, and if there is still tie, the one with the **smaller** `movie_j` should appear first. If there are fewer than 5 rented movies, then all of them should be returned. If no movies are currently being rented, then an empty list should be returned.

Implement the `MovieRentingSystem` class:

- `MovieRentingSystem(int n, int[][] entries)` Initializes the `MovieRentingSystem` object with `n` shops and the movies in `entries`.
- `List<Integer> search(int movie)` Returns a list of shops that have an **unrented copy** of the given `movie` as described above.
- `void rent(int shop, int movie)` Rents the given `movie` from the given `shop`.
- `void drop(int shop, int movie)` Drops off a previously rented `movie` at the given `shop`.
- `List<List<Integer>> report()` Returns a list of cheapest **rented** movies as described above.

Note: The test cases will be generated such that `rent` will only be called if the shop has an **unrented** copy of the movie, and `drop` will only be called if the shop had **previously rented** out the movie.

Example 1:

```
Input
["MovieRentingSystem", "search", "rent", "rent", "report", "drop", "search"]
[[3, [[0, 1, 5], [0, 2, 6], [0, 3, 7], [1, 1, 4], [1, 2, 7], [2, 1, 5]]], [1], [0, 1], [1, 2], [], [1, 2], [2]]
Output
[null, [1, 0, 2], null, null, [[0, 1], [1, 2]], null, [0, 1]]

Explanation
MovieRentingSystem movieRentingSystem = new MovieRentingSystem(3, [[0, 1, 5], [0, 2, 6], [0, 3, 7], [1, 1, 4],
movieRentingSystem.search(1);  // return [1, 0, 2], Movies of ID 1 are unrented at shops 1, 0, and 2. Shop 1 is
movieRentingSystem.rent(0, 1); // Rent movie 1 from shop 0. Unrented movies at shop 0 are now [2,3].
movieRentingSystem.rent(1, 2); // Rent movie 2 from shop 1. Unrented movies at shop 1 are now [1].
movieRentingSystem.report();   // return [[0, 1], [1, 2]]. Movie 1 from shop 0 is cheapest, followed by movie 2
movieRentingSystem.drop(1, 2); // Drop off movie 2 at shop 1. Unrented movies at shop 1 are now [1,2].
movieRentingSystem.search(2);  // return [0, 1]. Movies of ID 2 are unrented at shops 0 and 1. Shop 0 is cheape
```

Constraints:

- `1 <= n <= 3 * 10`^{5}
- `1 <= entries.length <= 10`^{5}
- `0 <= shop`$_i$ `< n`
- `1 <= movie`$_i$`, price`$_i$ `<= 10`^{4}
- Each shop carries **at most one** copy of a movie `movie`$_i$.
- At most `10`^{5} calls **in total** will be made to `search` , `rent` , `drop` and `report` .

JavaScript

```
const pr = console.log;
function MovieRentingSystem(n, entries) {
    let unrentM = new Map();
    let rentM = new Map();
    entries.map((x, i) => {
        let [shop, movie, price] = x;
        if (!unrentM.has(movie)) unrentM.set(movie, []);
        unrentM.get(movie).push([shop, movie, price]);
    });
    return { search, rent, drop, report }
    function search(movie) {
        let a = unrentM.get(movie) || [];
        a.sort((x, y) => {
            if (x[2] == y[2]) return x[0] - y[0];
            return x[2] - y[2];
        });
        return a.map(x => x[0]).slice(0, 5);
    }

    function rent(shop, movie) {
        let a = unrentM.get(movie) || [];
        let idx, cur;
        for (let i = 0; i < a.length; i++) {
            if (a[i][0] == shop) {
                idx = i;
                cur = a[i];
                break;
            }
        }
        if (cur) {
            if (!rentM.has(movie)) rentM.set(movie, []);
            rentM.get(movie).push(cur);
            a.splice(idx, 1);
            a.length == 0 ? unrentM.delete(movie) : unrentM.set(movie, a);
        }
    }

    function drop(shop, movie) {
        let a = rentM.get(movie) || [];
        let idx, cur;
        for (let i = 0; i < a.length; i++) {
            if (a[i][0] == shop) {
                idx = i;
                cur = a[i];
                break;
            }
        }
        if (cur) {
            if (!unrentM.has(movie)) unrentM.set(movie, []);
            unrentM.get(movie).push(cur);
            a.splice(idx, 1);
            a.length == 0 ? rentM.delete(movie) : rentM.set(movie, a);
        }
    }

    function report() {
        let d = [];
        for (const [, a] of rentM) {
            for (const e of a) {
                d.push(e);
            }
        }
        d.sort((x, y) => {
            if (x[2] != y[2]) return x[2] - y[2];
            if (x[0] != y[0]) return x[0] - y[0];
            return x[1] - y[1];
        });
        return d.map(x => [x[0], x[1]]).slice(0, 5);
    }
```

☐ **Custom Testcase** Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/513645437/) More Details (/submissions/detail/513645437/)

Share your acceptance!