6126. Design a Food Rating System

Design a food rating system that can do the following:

- **Modify** the rating of a food item listed in the system.
- Return the highest-rated food item for a type of cuisine in the system.

Implement the `FoodRatings` class:

- `FoodRatings(String[] foods, String[] cuisines, int[] ratings)` Initializes the system. The food items are described by `foods`, `cuisines` and `ratings`, all of which have a length of `n`.
  - `foods[i]` is the name of the ith food,
  - `cuisines[i]` is the type of cuisine of the ith food, and
  - `ratings[i]` is the initial rating of the ith food.
- `void changeRating(String food, int newRating)` Changes the rating of the food item with the name `food`.
- `String highestRated(String cuisine)` Returns the name of the food item that has the highest rating for the given type of `cuisine`. If there is a tie, return the item with the **lexicographically smaller** name.
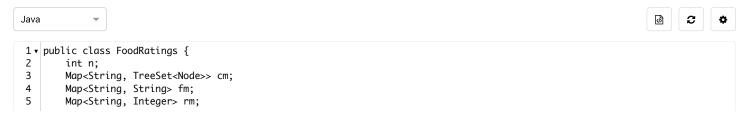
Note that a string `x` is lexicographically smaller than string `y` if `x` comes before `y` in dictionary order, that is, either `x` is a prefix of `y`, or if `i` is the first position such that `x[i] != y[i]`, then `x[i]` comes before `y[i]` in alphabetic order.

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

Example 1:

```
Input
["FoodRatings", "highestRated", "highestRated", "changeRating", "highestRated", "changeRating", "highestRated"]
[[["kimchi", "miso", "sushi", "moussaka", "ramen", "bulgogi"], ["korean", "japanese", "japanese", "greek", "japanese", "korean"
Output
[null, "kimchi", "ramen", null, "sushi", null, "ramen"]

Explanation
FoodRatings foodRatings = new FoodRatings(["kimchi", "miso", "sushi", "moussaka", "ramen", "bulgogi"], ["korean", "japanese", "
foodRatings.highestRated("korean"); // return "kimchi"
                                    // "kimchi" is the highest rated korean food with a rating of 9.
foodRatings.highestRated("japanese"); // return "ramen"
                                      // "ramen" is the highest rated japanese food with a rating of 14.
foodRatings.changeRating("sushi", 16); // "sushi" now has a rating of 16.
foodRatings.highestRated("japanese"); // return "sushi"
                                      // "sushi" is the highest rated japanese food with a rating of 16.
foodRatings.changeRating("ramen", 16); // "ramen" now has a rating of 16.
foodRatings.highestRated("japanese"); // return "ramen"
                                      // Both "sushi" and "ramen" have a rating of 16.
                                      // However, "ramen" is lexicographically smaller than "sushi".
```

Constraints:

- $1 <= n <= 2 * 10^4$
- `n == foods.length == cuisines.length == ratings.length`
- `1 <= foods[i].length, cuisines[i].length <= 10`
- `foods[i]`, `cuisines[i]` consist of lowercase English letters.
- $1 <= ratings[i] <= 10^8$
- All the strings in `foods` are **distinct**.
- `food` will be the name of a food item in the system across all calls to `changeRating`.
- `cuisine` will be a type of cuisine of **at least one** food item in the system across all calls to `highestRated`.
- At most $2 * 10^4$ calls **in total** will be made to `changeRating` and `highestRated`.

Java ▾

```java
1  public class FoodRatings {
2      int n;
3      Map<String, TreeSet<Node>> cm;
4      Map<String, String> fm;
5      Map<String, Integer> rm;
```

```
 6
 7 ▾      public FoodRatings(String[] foods, String[] cuisines, int[] ratings) {
 8            n = foods.length;
 9            cm = new HashMap<>(); // cuisine -> Node
10            fm = new HashMap<>(); // food -> cuisine
11            rm = new HashMap<>(); // food -> rating
12 ▾          for (int i = 0; i < n; i++) {
13                fm.put(foods[i], cuisines[i]);
14                rm.put(foods[i], ratings[i]);
15                cm.computeIfAbsent(cuisines[i], x -> new TreeSet<>()).add(new Node(ratings[i], foods[i]));
16            }
17        }
18
19 ▾      public void changeRating(String food, int newRating) {
20            String cuisine = fm.get(food);
21            cm.get(cuisine).remove(new Node(rm.get(food), food));
22            rm.put(food, newRating);
23            cm.get(cuisine).add(new Node(rm.get(food), food));
24        }
25
26 ▾      public String highestRated(String cuisine) {
27            return cm.get(cuisine).first().name;
28        }
29
30 ▾      class Node implements Comparable<Node> {
31            int rate;
32            String name;
33
34 ▾          Node(int rate, String name) {
35                this.rate = rate;
36                this.name = name;
37            }
38
39            @Override
40 ▾          public int compareTo(Node y) {
41                if (rate == y.rate) return name.compareTo(y.name); // same rating, lexical smaller comes first
42                return Integer.compare(y.rate, rate); // higher rating comes first
43            }
44        }
45  }
```

☐ **Custom Testcase**   (Use Example Testcases)

                                                                                      (▶ Run) (☁ Submit)

Submission Result: *Accepted* (/submissions/detail/755134075/) ❓   (More Details ❯ (/submissions/detail/755134075/))

Share your acceptance!

◁ 1