715. Range Module

DescriptionHintsSubmissionsDiscussSolution

A Range Module is a module that tracks ranges of numbers. Your task is to design and implement the following interfaces in an efficient manner.

- `addRange(int left, int right)` Adds the half-open interval `[left, right)`, tracking every real number in that interval. Adding an interval that partially overlaps with currently tracked numbers should add any numbers in the interval `[left, right)` that are not already tracked.
- `queryRange(int left, int right)` Returns true if and only if every real number in the interval `[left, right)` is currently being tracked.
- `removeRange(int left, int right)` Stops tracking every real number currently being tracked in the interval `[left, right)`.

Example 1:

```
addRange(10, 20): null
removeRange(14, 16): null
queryRange(10, 14): true (Every number in [10, 14) is being tracked)
queryRange(13, 15): false (Numbers like 14, 14.03, 14.17 in [13, 15) are not being tracked)
queryRange(16, 17): true (The number 16 in [16, 17) is still being tracked, despite the remove operation)
```

Note:

- A half open interval `[left, right)` denotes all real numbers `left <= x < right`.
- `0 < left < right < 10^9` in all calls to `addRange, queryRange, removeRange`.
- The total number of calls to `addRange` in a single test case is at most `1000`.
- The total number of calls to `queryRange` in a single test case is at most `5000`.
- The total number of calls to `removeRange` in a single test case is at most `1000`.

Seen this question in a real interview before?

```
class RangeModule {
public:
   void addRange(int left, int right) {
       int n = invals.size();
       vector<pair<int, int>> tmp;
       for (int i = 0; i <= n; i++) {
```

```
            if (i == n || invals[i].first > right) {
                tmp.push_back({left, right});
                while (i < n) tmp.push_back(invals[i++]);
            }
            else if (invals[i].second < left)
                tmp.push_back(invals[i]);
            else {
                left = min(left, invals[i].first);
                right = max(right, invals[i].second);
            }
        }
        swap(invals, tmp);
    }

    bool queryRange(int left, int right) {
        int n = invals.size(), l = 0, r = n-1;
        while (l <= r) {
            int m = l+(r-l)/2;
            if (invals[m].first >= right)
                r = m-1;
            else if (invals[m].second <= left)
                l = m+1;
            else
                return invals[m].first <= left && invals[m].second
>= right;
        }
        return false;
    }

    void removeRange(int left, int right) {
        int n = invals.size();
        vector<pair<int, int>> tmp;
        for (int i = 0; i < n; i++) {
            if (invals[i].second <= left || invals[i].first >=
right)
                tmp.push_back(invals[i]);
            else {
                if (invals[i].first < left)
tmp.push_back({invals[i].first, left});
                if (invals[i].second > right) tmp.push_back({right,
invals[i].second});
            }
        }
        swap(invals, tmp);
    }
private:
    vector<pair<int, int>> invals;
};
```