宫水三叶的刷题日记

线段树

Author ：宫水三叶
Date ：2021/10/07
QQ Group：703311589
WeChat ：oaoaya

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

噔噔噔噔，这是公众号「宫水三叶的刷题日记」的原创专题「线段树」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「线段树」即可获取最新下载链接。

💡**下面介绍使用本合集的最佳使用实践：**

学习算法：

1. 打开在线目录（Github 版 & Gitee 版）；
2. 从侧边栏的类别目录找到「线段树」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题‘
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流🍭🍭🍭

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **1109. 航班预订统计** ，难度为 **中等**。

Tag : 「区间求和问题」、「差分」、「线段树」

这里有 `n` 个航班，它们分别从 `1` 到 `n` 进行编号。

有一份航班预订表 `bookings` ，表中第 `i` 条预订记录 $bookings[i] = [first_i, last_i, seats_i]$ 意味着在从 $first_i$ 到 $last_i$ （包含 $first_i$ 和 $last_i$ ）的 每个航班 上预订了 $seats_i$ 个座位。

请你返回一个长度为 `n` 的数组 `answer`，其中 `answer[i]` 是航班 `i` 上预订的座位总数。

示例 1：

```
输入：bookings = [[1,2,10],[2,3,20],[2,5,25]], n = 5

输出：[10,55,45,25,25]

解释：
航班编号        1   2   3   4   5
预订记录 1 ：   10  10
预订记录 2 ：       20  20
预订记录 3 ：       25  25  25  25
总座位数：      10  55  45  25  25
因此，answer = [10,55,45,25,25]
```

示例 2：

```
输入：bookings = [[1,2,10],[2,2,15]], n = 2

输出：[10,25]

解释：
航班编号        1   2
预订记录 1 ：   10  10
预订记录 2 ：       15
总座位数：      10  25
因此，answer = [10,25]
```

提示：

- 1 <= n <= $2 * 10^4$
- 1 <= bookings.length <= $2 * 10^4$
- bookings[i].length == 3
- 1 <= firsti <= lasti <= n
- 1 <= seatsi <= 10^4

基本分析

本题只涉及「区间修改 + 单点查询」，属于「区间求和」问题中的入门难度。

对于各类「区间求和」问题，该用什么方式进行求解，之前在 这里 提到过。

此处可以再总结一下（加粗字体为最佳方案）：

- 数组不变，区间查询：**前缀和**、树状数组、线段树；
- 数组单点修改，区间查询：**树状数组**、线段树；
- 数组区间修改，单点查询：**差分**、线段树；
- 数组区间修改，区间查询：**线段树**。

> 注意：上述总结是对于一般性而言的（能直接解决的），对标的是模板问题。
> 但存在经过一些经过“额外”操作，对问题进行转化，从而使用别的解决方案求解的情况。
> 例如某些问题，我们可以先对原数组进行差分，然后使用树状数组，也能解决区间修改问题。
> 或者使用多个树状数组来维护多个指标，从而实现类似线段树的持久化标记操作。
> 但这些不属于一般性，所以就不添加到题解了。

差分

本题只涉及「区间修改 + 单点查询」，因此是一道「差分」的模板题。

「差分」可以看做是求「前缀和」的逆向过程。

对于一个「将区间 $[l, r]$ 整体增加一个值 v」操作，我们可以对差分数组 c 的影响看成两部分：

- 对 $c[l] += v$：由于差分是前缀和的逆向过程，这个操作对于将来的查询而言，带来的影响是对于所有的下标大于等于 l 的位置都增加了值 v；
- 对 $c[r + 1] -= v$：由于我们期望只对 $[l, r]$ 产生影响，因此需要对下标大于 r 的位置进行减值操作，从而抵消“影响”。

对于最后的构造答案，可看做是对每个下标做“单点查询”操作，只需要对差分数组求前缀和即可。

代码：

```
class Solution {
    public int[] corpFlightBookings(int[][] bs, int n) {
        int[] c = new int[n + 1];
        for (int[] bo : bs) {
            int l = bo[0] - 1, r = bo[1] - 1, v = bo[2];
            c[l] += v;
            c[r + 1] -= v;
        }
        int[] ans = new int[n];
        ans[0] = c[0];
        for (int i = 1; i < n; i++) {
            ans[i] = ans[i - 1] + c[i];
        }
        return ans;
    }
}
```

- 时间复杂度：令 `bs` 长度为 m，预处理差分数组的复杂度为 $O(m)$；构造答案复杂度为 $O(n)$。整体复杂度为 $O(m + n)$
- 空间复杂度：$O(n)$

线段树

在「基本分析」中，我们发现几乎所有的「区间求和」问题都可以使用线段树解决。

那么是否无脑写线段树呢？答案并不是，恰好相反。

线段树代码很长，且常数很大，实际表现不算很好。只有不得不写「线段树」的时候，我们才考虑线段树。

回到本题，由于涉及「区间修改」操作，因此我们需要对线段树进行持久化标记（懒标记），从而确保操作仍为 $\log$ 级别的复杂度。

代码：

```java
class Solution {
    class Node {
        int l, r, v, add;
        Node(int _l, int _r) {
            l = _l; r = _r;
        }
    }
    int N = 20009;
    Node[] tr = new Node[N * 4];
    void pushup(int u) {
        tr[u].v = tr[u << 1].v + tr[u << 1 | 1].v;
    }
    void pushdown(int u) {
        int add = tr[u].add;
        tr[u << 1].v += add;
        tr[u << 1].add += add;
        tr[u << 1 | 1].v += add;
        tr[u << 1 | 1].add += add;
        tr[u].add = 0;
    }
    void build(int u, int l, int r) {
        tr[u] = new Node(l, r);
        if (l != r) {
            int mid = l + r >> 1;
            build(u << 1, l, mid);
            build(u << 1 | 1, mid + 1, r);
        }
    }
    void update(int u, int l, int r, int v) {
        if (l <= tr[u].l && tr[u].r <= r) {
            tr[u].v += v;
            tr[u].add += v;
        } else {
            pushdown(u);
            int mid = tr[u].l + tr[u].r >> 1;
            if (l <= mid) update(u << 1, l, r, v);
            if (r > mid) update(u << 1 | 1, l, r, v);
            pushup(u);
        }
    }
    int query(int u, int l, int r) {
        if (l <= tr[u].l && tr[u].r <= r) {
            return tr[u].v;
        } else {
            pushdown(u);
            int mid = tr[u].l + tr[u].r >> 1;
```

```
            int ans = 0;
            if (l <= mid) ans += query(u << 1, l, r);
            if (r > mid) ans += query(u << 1 | 1, l, r);
            return ans;
        }
    }
    public int[] corpFlightBookings(int[][] bs, int n) {
        build(1, 1, n);
        for (int[] bo : bs) {
            update(1, bo[0], bo[1], bo[2]);
        }
        int[] ans = new int[n];
        for (int i = 0; i < n; i++) {
            ans[i] = query(1, i + 1, i + 1);
        }
        return ans;
    }
}
```

- 时间复杂度：线段树建树复杂度为 $O(n)$，其余操作复杂度为 $O(\log n)$。对于本题，令 `bs` 长度为 m，整体复杂度为 $O(m \log n + n \log n)$
- 空间复杂度：$O(n)$

> **🍭 更多精彩内容，欢迎关注：**公众号** / Github / LeetCode / **知乎** **

题目描述

这是 LeetCode 上的 **1893. 检查是否区域内所有整数都被覆盖** ，难度为 **简单**。

Tag : 「模拟」、「树状数组」、「线段树」

给你一个二维整数数组 ranges 和两个整数 left 和 right 。每个 ranges[i] = [starti, endi] 表示一个从 starti 到 endi 的 闭区间 。

如果闭区间 [left, right] 内每个整数都被 ranges 中 至少一个 区间覆盖，那么请你返回 true ，否则返回 false 。

已知区间 ranges[i] = [starti, endi] ，如果整数 x 满足 starti <= x <= endi ，那么我们称整数x 被覆盖了。

示例 1：

```
输入：ranges = [[1,2],[3,4],[5,6]], left = 2, right = 5

输出：true

解释：2 到 5 的每个整数都被覆盖了：
- 2 被第一个区间覆盖。
- 3 和 4 被第二个区间覆盖。
- 5 被第三个区间覆盖。
```

示例 2：

```
输入：ranges = [[1,10],[10,20]], left = 21, right = 21

输出：false

解释：21 没有被任何一个区间覆盖。
```

提示：

- 1 <= ranges.length <= 50
- 1 <= starti <= endi <= 50
- 1 <= left <= right <= 50

模拟

一个简单的想法是根据题意进行模拟，检查 $[left, right]$ 中的每个整数，如果检查过程中发现某个整数没被 $ranges$ 中的闭区间所覆盖，那么直接返回 `False`，所有数值通过检查则返回 `True`。

代码：

```java
class Solution {
    public boolean isCovered(int[][] rs, int l, int r) {
        for (int i = l; i <= r; i++) {
            boolean ok = false;
            for (int[] cur : rs) {
                int a = cur[0], b = cur[1];
                if (a <= i && i <= b) {
                    ok = true;
                    break;
                }
            }
            if (!ok) return false;
        }
        return true;
    }
}
```

- 时间复杂度：令 $[left, right]$ 之间整数数量为 n，$ranges$ 长度为 m。整体复杂度为 $O(n * m)$
- 空间复杂度：$O(1)$

树状数组

针对此题，可以有一个很有意思的拓展，将本题难度提升到【中等】甚至是【困难】。

将查询 $[left, right]$ 修改为「四元查询数组」$querys$，每个 $querys[i]$ 包含四个指标 (a, b, l, r)：代表询问 $[l, r]$ 中的每个数是否在 $range$ 中 $[a, b]$ 的闭区间所覆盖过。

如果进行这样的拓展的话，那么我们需要使用「持久化树状数组」或者「主席树」来配合「容斥原理」来做。

基本思想都是使用 $range[0, b]$ 的计数情况减去 $range[0, a-1]$ 的计数情况来得出 $[a, b]$ 的计数情况。

回到本题，由于数据范围很小，只有 50，我们可以使用「树状数组」进行求解：

- `void add(int x, int u)`：对于数值 x 出现次数进行 $+u$ 操作；
- `int query(int x)`：查询某个满足 $<= x$ 的数值的个数。

那么显然，如果我们需要查询一个数值 x 是否出现过，可以通过查询 $cnt = query(x) -$

$query(x-1)$ 来得知。

代码：

```
class Solution {
    int n = 55;
    int[] tr = new int[n];
    int lowbit(int x) {
        return x & -x;
    }
    void add(int x, int u) {
        for (int i = x; i <= n; i += lowbit(i)) tr[i] += u;
    }
    int query(int x) {
        int ans = 0;
        for (int i = x; i > 0; i -= lowbit(i)) ans += tr[i];
        return ans;
    }
    public boolean isCovered(int[][] rs, int l, int r) {
        for (int[] cur : rs) {
            int a = cur[0], b = cur[1];
            for (int i = a; i <= b; i++) {
                add(i, 1);
            }
        }
        for (int i = l; i <= r; i++) {
            int cnt = query(i) - query(i - 1);
            if (cnt == 0) return false;
        }
        return true;
    }
}
```

- 时间复杂度：令 $[left, right]$ 之间整数数量为 n，$\sum_{i=0}^{range.legth-1} ranges[i].length$ 为 sum，常数 C 固定为 55。建树复杂度为 $O(sum \log C)$，查询查询复杂度为 $O(n \log C)$。整体复杂度为 $O(sum \log C + n \log C)$
- 空间复杂度：$O(C)$

树状数组（去重优化）

在朴素的「树状数组」解法中，我们无法直接查询 $[l, r]$ 区间中被覆盖过的个数的根本原因是「某个值可能会被重复添加到树状数组中」。

因此，一种更加优秀的做法：**在往树状数组中添数的时候进行去重，然后通过 $cnt = query(r) - query(l-1)$ 直接得出 $[l, r]$ 范围内有多少个数被添加过。**

这样的 `Set` 去重操作可以使得我们查询的复杂度从 $O(n \log C)$ 下降到 $O(\log C)$。

> 由于数值范围很小，自然也能够使用数组来代替 `Set` 进行标记（见 P2）

代码：

```
class Solution {
    int n = 55;
    int[] tr = new int[n];
    int lowbit(int x) {
        return x & -x;
    }
    void add(int x, int u) {
        for (int i = x; i <= n; i += lowbit(i)) tr[i] += u;
    }
    int query(int x) {
        int ans = 0;
        for (int i = x; i > 0; i -= lowbit(i)) ans += tr[i];
        return ans;
    }
    public boolean isCovered(int[][] rs, int l, int r) {
        Set<Integer> set = new HashSet<>();
        for (int[] cur : rs) {
            int a = cur[0], b = cur[1];
            for (int i = a; i <= b; i++) {
                if (!set.contains(i)) {
                    add(i, 1);
                    set.add(i);
                }
            }
        }
        int tot = r - l + 1, cnt = query(r) - query(l - 1);
        return tot == cnt;
    }
}
```

```java
class Solution {
    int n = 55;
    int[] tr = new int[n];
    boolean[] vis = new boolean[n];
    int lowbit(int x) {
        return x & -x;
    }
    void add(int x, int u) {
        for (int i = x; i <= n; i += lowbit(i)) tr[i] += u;
    }
    int query(int x) {
        int ans = 0;
        for (int i = x; i > 0; i -= lowbit(i)) ans += tr[i];
        return ans;
    }
    public boolean isCovered(int[][] rs, int l, int r) {
        for (int[] cur : rs) {
            int a = cur[0], b = cur[1];
            for (int i = a; i <= b; i++) {
                if (!vis[i]) {
                    add(i, 1);
                    vis[i] = true;
                }
            }
        }
        int tot = r - l + 1, cnt = query(r) - query(l - 1);
        return tot == cnt;
    }
}
```

- 时间复杂度：令 $[left, right]$ 之间整数数量为 n，$\sum_{i=0}^{range.legth-1} ranges[i].length$ 为 sum，常数 C 固定为 55。建树复杂度为 $O(sum \log C)$，查询查询复杂度为 $O(\log C)$。整体复杂度为 $O(sum \log C + \log C)$
- 空间复杂度：$O(C + \sum_{i=0}^{range.legth-1} ranges[i].length)$

线段树（不含“懒标记”）

更加进阶的做法是使用「线段树」来做，与「树状数组（优化）」解法一样，线段树配合持久化也可以用于求解「在线」问题。

与主要解决「单点修改 & 区间查询」的树状数组不同，线段树能够解决绝大多数「区间修改（区间修改/单点修改）& 区间查询」问题。

对于本题，由于数据范围只有 55，因此我们可以使用与「树状数组（优化）」解法相同的思路，实现一个不包含“懒标记”的线段树来做（仅支持单点修改 & 区间查询）。

代码：

```java
class Solution {
    // 代表 [l, r] 区间有 cnt 个数被覆盖
    class Node {
        int l, r, cnt;
        Node (int _l, int _r, int _cnt) {
            l = _l; r = _r; cnt = _cnt;
        }
    }
    int N = 55;
    Node[] tr = new Node[N * 4];
    void pushup(int u) {
        tr[u].cnt = tr[u << 1].cnt + tr[u << 1 | 1].cnt;
    }
    void build(int u, int l, int r) {
        if (l == r) {
            tr[u] = new Node(l, r, 0);
        } else {
            tr[u] = new Node(l, r, 0);
            int mid = l + r >> 1;
            build(u << 1, l, mid);
            build(u << 1 | 1, mid + 1, r);
            pushup(u);
        }
    }
    // 从 tr 数组的下标 u 开始，在数值 x 的位置进行标记
    void update(int u, int x) {
        if (tr[u].l == x && tr[u].r == x) {
            tr[u].cnt = 1;
        } else {
            int mid = tr[u].l + tr[u].r >> 1;
            if (x <= mid) update(u << 1, x);
            else update(u << 1 | 1, x);
            pushup(u);
        }
    }
    // 从 tr 数组的下标 u 开始，查询 [l,r] 范围内有多少个数值被标记
    int query(int u, int l, int r) {
        if (l <= tr[u].l && tr[u].r <= r) return tr[u].cnt;
        int mid = tr[u].l + tr[u].r >> 1;
        int ans = 0;
        if (l <= mid) ans += query(u << 1, l, r);
        if (r > mid) ans += query(u << 1 | 1, l, r);
        return ans;
    }
    public boolean isCovered(int[][] rs, int l, int r) {
        build(1, 1, N);
```

```
        for (int[] cur : rs) {
            int a = cur[0], b = cur[1];
            for (int i = a; i <= b; i++) {
                update(1, i);
            }
        }
        int tot = r - l + 1 , cnt = query(1, l, r);
        return tot == cnt;
    }
}
```

- 时间复杂度：令 $[left, right]$ 之间整数数量为 n，$\sum_{i=0}^{range.legth-1} ranges[i].length$ 为 sum，常数 C 固定为 55。建树复杂度为 $O(sum \log C)$，查询查询复杂度为 $O(\log C)$。整体复杂度为 $O(sum \log C + \log C)$
- 空间复杂度：$O(C * 4)$

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

💡**更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。**

最新专题合集资料下载，可关注公众号「宫水三叶的刷题日记」，回台回复「线段树」获取下载链接。

觉得专题不错，可以请作者吃糖 🍭🍭🍭 ：


“给作者手机充个电”
YOLO 的赞赏码