

宫水三叶的刷题日记

差分

Author : 宫水三叶

Date : 2021/10/07

QQ Group: 703311589

WeChat : oaoaya

宫水三叶

刷题日记

公众号: 宫水三叶的刷题日记

噔噔噔噔，这是公众号「[宫水三叶的刷题日记](#)」的原创专题「差分」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「差分」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「差分」；
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   

题目描述

这是 LeetCode 上的 [995. K 连续位的最小翻转次数](#)，难度为 困难。

Tag：「贪心」、「差分」

在仅包含 0 和 1 的数组 A 中，一次 K 位翻转包括选择一个长度为 K 的（连续）子数组，同时将子数组中的每个 0 更改为 1，而每个 1 更改为 0。

返回所需的 K 位翻转的最小次数，以便数组没有值为 0 的元素。如果不可能，返回 -1。

示例 1：

宫水三叶
刷题日记

公众号: 宫水三叶的刷题日记

输入：A = [0,1,0], K = 1
输出：2
解释：先翻转 A[0]，然后翻转 A[2]。

示例 2：

输入：A = [1,1,0], K = 2
输出：-1
解释：无论我们怎样翻转大小为 2 的子数组，我们都不能使数组变为 [1,1,1]。

示例 3：

输入：A = [0,0,0,1,0,1,1,0], K = 3
输出：3
解释：
翻转 A[0],A[1],A[2]：A变成 [1,1,1,1,0,1,1,0]
翻转 A[4],A[5],A[6]：A变成 [1,1,1,1,1,0,0,0]
翻转 A[5],A[6],A[7]：A变成 [1,1,1,1,1,1,1,1]

提示：

- $1 \leq A.length \leq 30000$
- $1 \leq K \leq A.length$

贪心解法

目标是将数组的每一位都变为 1，因此对于每一位 0 都需要翻转。

我们可以从前往后处理，遇到 0 则对后面的 k 位进行翻转。

这样我们的算法复杂度是 $O(nk)$ 的，数据范围是 3w（数量级为 10^4 ），极限数据下单秒的运算量在 10^8 以上，会有超时风险。

宫水三叶
の
刷题日记

公众号：宫水三叶的刷题日记

```

class Solution {
    public int minKBitFlips(int[] nums, int k) {
        int n = nums.length;
        int ans = 0;
        for (int i = 0; i < n; i++) {
            if (nums[i] == 0) {
                if (i + k > n) return -1;
                for (int j = i; j < i + k; j++) nums[j] ^= 1;
                ans++;
            }
        }
        return ans;
    }
}

```

- 时间复杂度： $O(nk)$
- 空间复杂度： $O(1)$

补充

评论有同学提出了一些有价值的疑问，我觉得挺有代表性的，因此补充到题解：

1. 为什么这样的解法是「贪心解法」，而不是「暴力解法」？

首先「暴力解法」必然是对所有可能出现的翻转方案进行枚举，然后检查每一个方案得到的结果是否符合全是 1 的要求。

这样的解法，才是暴力解法，它的本质是通过「穷举」找答案。复杂度是指数级别的。

而我们的「朴素贪心解法」只是执行了众多翻转方案中的一种。

举个 🍌，对于 `nums = [0,0,1,1]` 并且 `k = 2` 的数据：

暴力解法应该是「枚举」以下三种方案：

1. 只翻转以第一个 0 开头的子数组（长度固定为 2）
2. 只翻转以第二个 0 开头的子数组（长度固定为 2）
3. 同时翻转第一个 0 开头和第二个 0 开头的子数组（长度固定为 2，只不过这时候第一个 0 被翻转了一次，第二个 0 被翻转了两次）

然后对三种方案得到的最终解进行检查，找出符合结果全是 1 的方案。这种通过「穷举」方案检查合法性的解法才是「暴力」解法。

2. 为什么我采用了与「朴素贪心」解法相似的做法，超时了？

结果测试 C++、Python 超时，只有 Java 能过。

同样是 97 号样例数据，提交给 LeetCode 执行。Java 运行 200 ms 以内，而 C++ 运行 600 ms。

贪心 + 差分解法

由于我们总是对连续的一段进行「相同」的操作，同时只有「奇数」次数的翻转才会真正改变当前位置上的值。

自然而然，我们会想到使用数组 `arr` 来记录每一位的翻转次数。

同时我们又不希望是通过「遍历记 `arr` 的 `k` 位进行 +1」来完成统计。

因此可以使用差分数组来进行优化：当需要对某一段 `[l, r]` 进行 +1 的时候，只需要 `arr[l]++` 和 `arr[r + 1]--` 即可。

```
class Solution {
    public int minKBitFlips(int[] nums, int k) {
        int n = nums.length;
        int ans = 0;
        int[] arr = new int[n + 1];
        for (int i = 0, cnt = 0; i < n; i++) {
            cnt += arr[i];
            if ((nums[i] + cnt) % 2 == 0) {
                if (i + k > n) return -1;
                arr[i + 1]++;
                arr[i + k]--;
                ans++;
            }
        }
        return ans;
    }
}
```

宫水三叶
刷题日记

公众号：宫水三叶的刷题日记

- 时间复杂度： $O(n)$
- 空间复杂度： $O(n)$

证明

为什么「一遇到 0 就马上进行翻转」这样的做法得到的是最优解？

这道题的贪心证明思路和 [765. 情侣牵手](#) 是一样的。

本质上是在证明当我在处理第 k 个位置的 0 的时候，前面 $k - 1$ 个位置不存在 0，接下来要如何进行操作，可使得总的翻转次数最小。

如果你上次真正理解了我的证明过程的话，那么你会很容易就能证明出本题的贪心思路。

所以这次将这个证明过程留给大家思考～

为什么要「证明」或「理解证明」？

证明的意义在于，你知道为什么这样做是对的。

带来的好处是：

1. 一道「贪心」题目能搞清楚证明，那么同类的「贪心」题目你就都会做了。否则就会停留在“我知道这道题可以这样贪心，别的题我不确定是否也能这样做”。
2. 在「面试」阶段，你可以很清晰讲解你的思路。让面试官从你的「思维方式」上喜欢上你

更多与证明/分析相关的题解：

[561. 数组拆分 I](#)：反证法证明贪心算法的正确性

[765. 情侣牵手](#)：为什么交换任意一个都是对的？：两种 100% 的解法：并查集 & 贪心

[1579. 保证图可完全遍历](#)：为什么先处理公共边是对的？含贪心证明 + 数组模板～

1631. 最小体力消耗路径：反证法证明思路的合法性

11. 盛最多水的容器：双指针+贪心解法【含证明】

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， <code>answer =</code>	<code>[10,55,45,25,25]</code>				

示例 2：

宫水三叶
の
刷题日记

公众号：宫水三叶的刷题日记

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：

- $1 \leq n \leq 2 * 10^4$
- $1 \leq \text{bookings.length} \leq 2 * 10^4$
- $\text{bookings}[i].\text{length} == 3$
- $1 \leq \text{firsti} \leq \text{lasti} \leq n$
- $1 \leq \text{seatsi} \leq 10^4$

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： $O(n)$

刷题日记

公众号: 宫水三叶的刷题日记

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：

宫水三叶
の
刷题日记

公众号：宫水三叶的刷题日记

```

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](#) **

💡 **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](#)」，后台回复「差分」获取下载链接。

觉得专题不错，可以请作者吃糖 🍬🍬🍬：

宫水三叶
の
刷题日记

公众号: 宫水三叶的刷题日记



“给作者手机充个电”

YOLO 的赞赏码

版权声明：任何形式的转载请保留出处 [Wiki](#)。