小而美的算法技巧：差分数组

通知：数据结构精品课持续更新中，详情见这里。

读完本文，你不仅学会了算法套路，还可以顺便解决如下题目：

牛客	LeetCode	力扣	难度
-	370. Range Addition🔒	370. 区间加法🔒	🟠
-	1094. Car Pooling	1094. 拼车	🟠
-	1109. Corporate Flight Bookings	1109. 航班预订统计	🟠

———

前文 前缀和技巧详解 写过的前缀和技巧是非常常用的算法技巧，**前缀和主要适用的场景是原始数组不会被修改的情况下，频繁查询某个区间的累加和。**

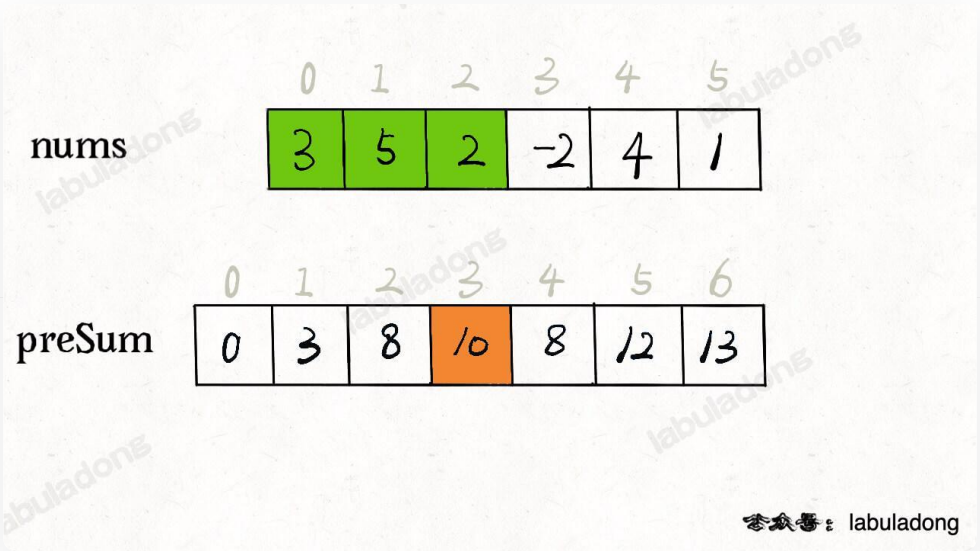
没看过前文没关系，这里简单介绍一下前缀和，核心代码就是下面这段：

```
class PrefixSum {
    // 前缀和数组
    private int[] prefix;

    /* 输入一个数组，构造前缀和 */
    public PrefixSum(int[] nums) {
        prefix = new int[nums.length + 1];
        // 计算 nums 的累加和
```

```
        for (int i = 1; i < prefix.length; i++) {
            prefix[i] = prefix[i - 1] + nums[i - 1];
        }
    }

    /* 查询闭区间 [i, j] 的累加和 */
    public int query(int i, int j) {
        return prefix[j + 1] - prefix[i];
    }
}
```



`prefix[i]` 就代表着 `nums[0..i-1]` 所有元素的累加和，如果我们想求区间 `nums[i..j]` 的累加和，只要计算 `prefix[j+1] - prefix[i]` 即可，而不需要遍历整个区间求和。

本文讲一个和前缀和思想非常类似的算法技巧「差分数组」，**差分数组的主要适用场景是频繁对原始数组的某个区间的元素进行增减**。

比如说，我给你输入一个数组 `nums`，然后又要求给区间 `nums[2..6]` 全部加 1，再给 `nums[3..9]` 全部减 3，再给 `nums[0..4]` 全部加 2，再给...

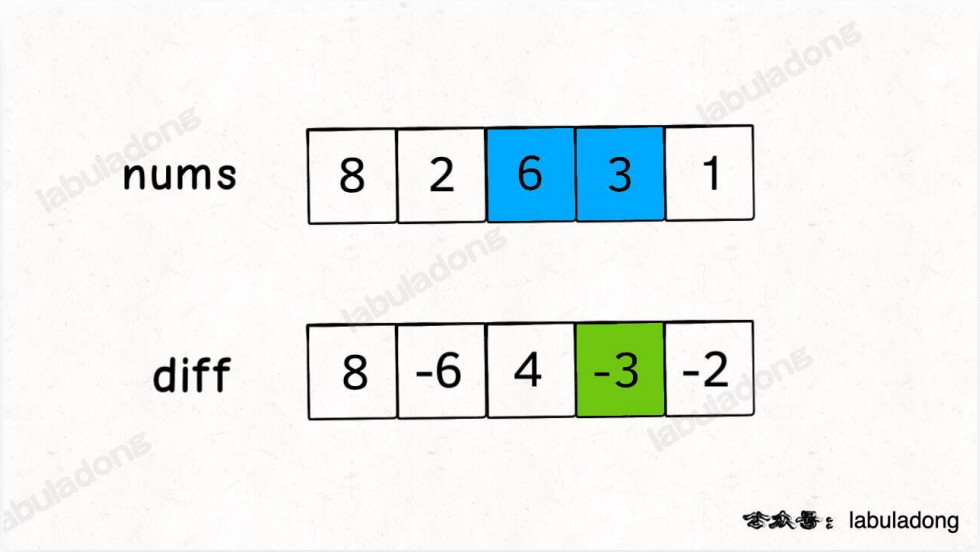
一通操作猛如虎，然后问你，最后 `nums` 数组的值是什么？

常规的思路很容易，你让我给区间 `nums[i..j]` 加上 `val`，那我就一个 for 循环给它们都加上呗，还能咋样？这种思路的时间复杂度是 O(N)，由于这个场景下对 `nums` 的修改非常频繁，所以效率会很低下。

这里就需要差分数组的技巧，类似前缀和技巧构造的 `prefix` 数组，我们先对 `nums` 数组构造一个

`diff` 差分数组，**`diff[i]` 就是 `nums[i]` 和 `nums[i-1]` 之差**：

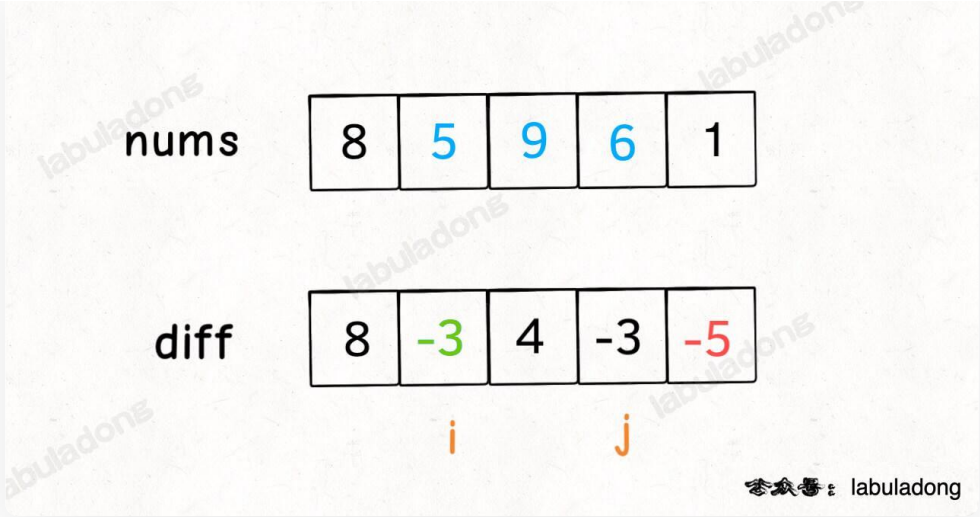
```
int[] diff = new int[nums.length];
// 构造差分数组
diff[0] = nums[0];
for (int i = 1; i < nums.length; i++) {
    diff[i] = nums[i] - nums[i - 1];
}
```



通过这个 `diff` 差分数组是可以反推出原始数组 `nums` 的，代码逻辑如下：

```
int[] res = new int[diff.length];
// 根据差分数组构造结果数组
res[0] = diff[0];
for (int i = 1; i < diff.length; i++) {
    res[i] = res[i - 1] + diff[i];
}
```

这样构造差分数组 `diff`，就可以快速进行区间增减的操作，如果你想对区间 `nums[i..j]` 的元素全部加 3，那么只需要让 `diff[i] += 3`，然后再让 `diff[j+1] -= 3` 即可：



原理很简单，回想 `diff` 数组反推 `nums` 数组的过程，`diff[i] += 3` 意味着给 `nums[i..]` 所有的元素都加了 3，然后 `diff[j+1] -= 3` 又意味着对于 `nums[j+1..]` 所有元素再减 3，那综合起来，是不是就是对 `nums[i..j]` 中的所有元素都加 3 了？

只要花费 O(1) 的时间修改 `diff` 数组，就相当于给 `nums` 的整个区间做了修改。多次修改 `diff`，然后通过 `diff` 数组反推，即可得到 `nums` 修改后的结果。

现在我们把差分数组抽象成一个类，包含 `increment` 方法和 `result` 方法：

```
// 差分数组工具类
class Difference {
    // 差分数组
    private int[] diff;

    /* 输入一个初始数组，区间操作将在这个数组上进行 */
    public Difference(int[] nums) {
        assert nums.length > 0;
        diff = new int[nums.length];
        // 根据初始数组构造差分数组
        diff[0] = nums[0];
        for (int i = 1; i < nums.length; i++) {
            diff[i] = nums[i] - nums[i - 1];
        }
    }

    /* 给闭区间 [i, j] 增加 val（可以是负数）*/
    public void increment(int i, int j, int val) {
        diff[i] += val;
        if (j + 1 < diff.length) {
            diff[j + 1] -= val;
```

```
        }
    }

    /* 返回结果数组 */
    public int[] result() {
        int[] res = new int[diff.length];
        // 根据差分数组构造结果数组
        res[0] = diff[0];
        for (int i = 1; i < diff.length; i++) {
            res[i] = res[i - 1] + diff[i];
        }
        return res;
    }
}
```

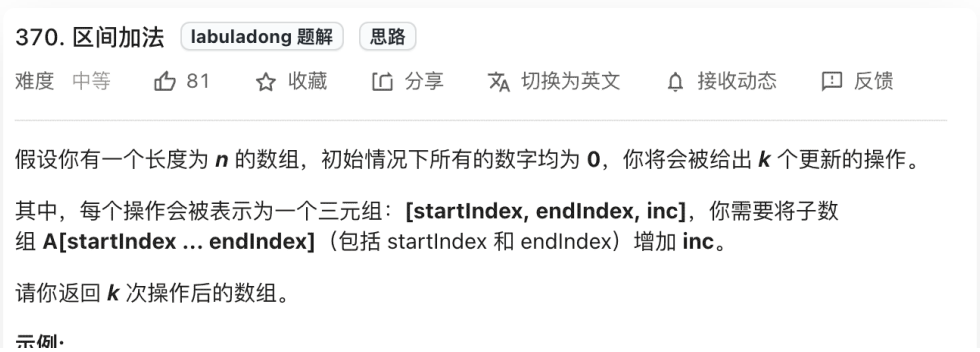
这里注意一下 `increment` 方法中的 if 语句：

```
public void increment(int i, int j, int val) {
    diff[i] += val;
    if (j + 1 < diff.length) {
        diff[j + 1] -= val;
    }
}
```

当 `j+1 >= diff.length` 时，说明是对 `nums[i]` 及以后的整个数组都进行修改，那么就不需要再给 `diff` 数组减 `val` 了。

算法实践

首先，力扣第 370 题「区间加法」就直接考察了差分数组技巧：



370. 区间加法 labuladong 题解 思路

难度 中等 81 收藏 分享 切换为英文 接收动态 反馈

假设你有一个长度为 ***n*** 的数组，初始情况下所有的数字均为 **0**，你将会被给出 ***k*** 个更新的操作。

其中，每个操作会被表示为一个三元组：**[startIndex, endIndex, inc]**，你需要将子数组 **A[startIndex ... endIndex]**（包括 startIndex 和 endIndex）增加 **inc**。

请你返回 ***k*** 次操作后的数组。

示例:

输入： length = 5, updates = [[1,3,2],[2,4,3],[0,2,-2]]
输出： [-2,0,3,5,3]

解释：

```
初始状态：
[0,0,0,0,0]

进行了操作 [1,3,2] 后的状态：
[0,2,2,2,0]

进行了操作 [2,4,3] 后的状态：
[0,2,5,5,3]

进行了操作 [0,2,-2] 后的状态：
[-2,0,3,5,3]
```

那么我们直接复用刚才实现的 `Difference` 类就能把这道题解决掉：

```
int[] getModifiedArray(int length, int[][] updates) {
    // nums 初始化为全 0
    int[] nums = new int[length];
    // 构造差分解法
    Difference df = new Difference(nums);

    for (int[] update : updates) {
        int i = update[0];
        int j = update[1];
        int val = update[2];
        df.increment(i, j, val);
    }

    return df.result();
}
```

当然，实际的算法题可能需要我们对题目进行联想和抽象，不会这么直接地让你看出来要用差分数组技巧，这里看一下力扣第 1109 题「航班预订统计」：

这里有 `n` 个航班，它们分别从 `1` 到 `n` 进行编号。

我们这儿有一份航班预订表，表中第 `i` 条预订记录 `bookings[i] = [i, j, k]` 意味着我们在从 `i` 到 `j` 的每个航班上预订了 `k` 个座位。

请你返回一个长度为 `n` 的数组 `answer`，按航班编号顺序返回每个航班上预订的座位数。

示例：

输入： bookings = [[1,2,10],[2,3,20],[2,5,25]], n = 5
输出： [10,55,45,25,25]

函数签名如下：

```
int[] corpFlightBookings(int[][] bookings, int n)
```

这个题目就在那绕弯弯，其实它就是个差分数组的题，我给你翻译一下：

给你输入一个长度为 `n` 的数组 `nums`，其中所有元素都是 0。再给你输入一个 `bookings`，里面是若干三元组 `(i, j,k)`，每个三元组的含义就是要求你给 `nums` 数组的闭区间 `[i-1,j-1]` 中所有元素都加上 `k`。请你返回最后的 `nums` 数组是多少？

> PS：因为题目说的 `n` 是从 1 开始计数的，而数组索引从 0 开始，所以对于输入的三元组 `(i, j,k)`，数组区间应该对应 `[i-1,j-1]`。

这么一看，不就是一道标准的差分数组题嘛？我们可以直接复用刚才写的类：

```
int[] corpFlightBookings(int[][] bookings, int n) {
    // nums 初始化为全 0
    int[] nums = new int[n];
    // 构造差分解法
    Difference df = new Difference(nums);

    for (int[] booking : bookings) {
        // 注意转成数组索引要减一哦
        int i = booking[0] - 1;
        int j = booking[1] - 1;
        int val = booking[2];
        // 对区间 nums[i..j] 增加 val
        df.increment(i, j, val);
```

```
        }
        // 返回最终的结果数组
        return df.result();
    }
```

这道题就解决了。

还有一道很类似的题目是力扣第 1094 题「拼车」，我简单描述下题目：

你是一个开公交车的司机，公交车的最大载客量为 `capacity`，沿途要经过若干车站，给你一份乘客行程表 `int[][] trips`，其中 `trips[i] = [num, start, end]` 代表着有 `num` 个旅客要从站点 `start` 上车，到站点 `end` 下车，请你计算是否能够一次把所有旅客运送完毕（不能超过最大载客量 `capacity`）。

函数签名如下：

```
boolean carPooling(int[][] trips, int capacity);
```

比如输入：

```
trips = [[2,1,5],[3,3,7]], capacity = 4
```

这就不能一次运完，因为 `trips[1]` 最多只能上 2 人，否则车就会超载。

相信你已经能够联想到差分数组技巧了：**`trips[i]` 代表着一组区间操作，旅客的上车和下车就相当于数组的区间加减；只要结果数组中的元素都小于 `capacity`，就说明可以不超载运输所有旅客。**

但问题是，差分数组的长度（车站的个数）应该是多少呢？题目没有直接给，但给出了数据取值范围：

```
0 <= trips[i][1] < trips[i][2] <= 1000
```

车站编号从 0 开始，最多到 1000，也就是最多有 1001 个车站，那么我们的差分数组长度可以直接设置为 1001，这样索引刚好能够涵盖所有车站的编号：

```
boolean carPooling(int[][] trips, int capacity) {
    // 最多有 1001 个车站
    int[] nums = new int[1001];
    // 构造差分解法
    Difference df = new Difference(nums);

    for (int[] trip : trips) {
        // 乘客数量
        int val = trip[0];
        // 第 trip[1] 站乘客上车
        int i = trip[1];
        // 第 trip[2] 站乘客已经下车，
        // 即乘客在车上的区间是 [trip[1], trip[2] - 1]
        int j = trip[2] - 1;
        // 进行区间操作
        df.increment(i, j, val);
    }

    int[] res = df.result();

    // 客车自始至终都不应该超载
    for (int i = 0; i < res.length; i++) {
        if (capacity < res[i]) {
            return false;
        }
    }
    return true;
}
```

至此，这道题也解决了。

最后，差分数组和前缀和数组都是比较常见且巧妙的算法技巧，分别适用不同的场景，而且是会者不难，难者不会。所以，关于差分数组的使用，你学会了吗？

> 最后打个广告，我亲自制作了一门 数据结构精品课，以视频课为主，手把手带你实现常用的数据结构及相关算法，旨在帮助算法基础较为薄弱的读者深入理解常用数据结构的底层原理，在算法学习中少走弯路。