前缀和技巧

相关推荐：

- 如何去除有序数组的重复元素
- 区间调度之区间合并问题

读完本文，你不仅学会了算法套路，还可以顺便去 LeetCode 上拿下如下题目：

560.和为K的子数组

今天来聊一道简单却十分巧妙的算法问题：算出一共有几个和为 `k` 的子数组。

> 给定一个整数数组和一个整数 **k**，你需要找到该数组中和为 **k** 的连续的子数组的个数。
>
> **示例 1：**
>
> ```
> 输入:nums = [1,1,1], k = 2
> 输出: 2 , [1,1] 与 [1,1] 为两种不同的情况。
> ```
>
> **说明：**
>
> 1. 数组的长度为 [1, 20,000]。
> 2. 数组中元素的范围是 [-1000, 1000] ，且整数 **k** 的范围是 [-1e7, 1e7]。

那我把所有子数组都穷举出来，算它们的和，看看谁的和等于 `k` 不就行了。

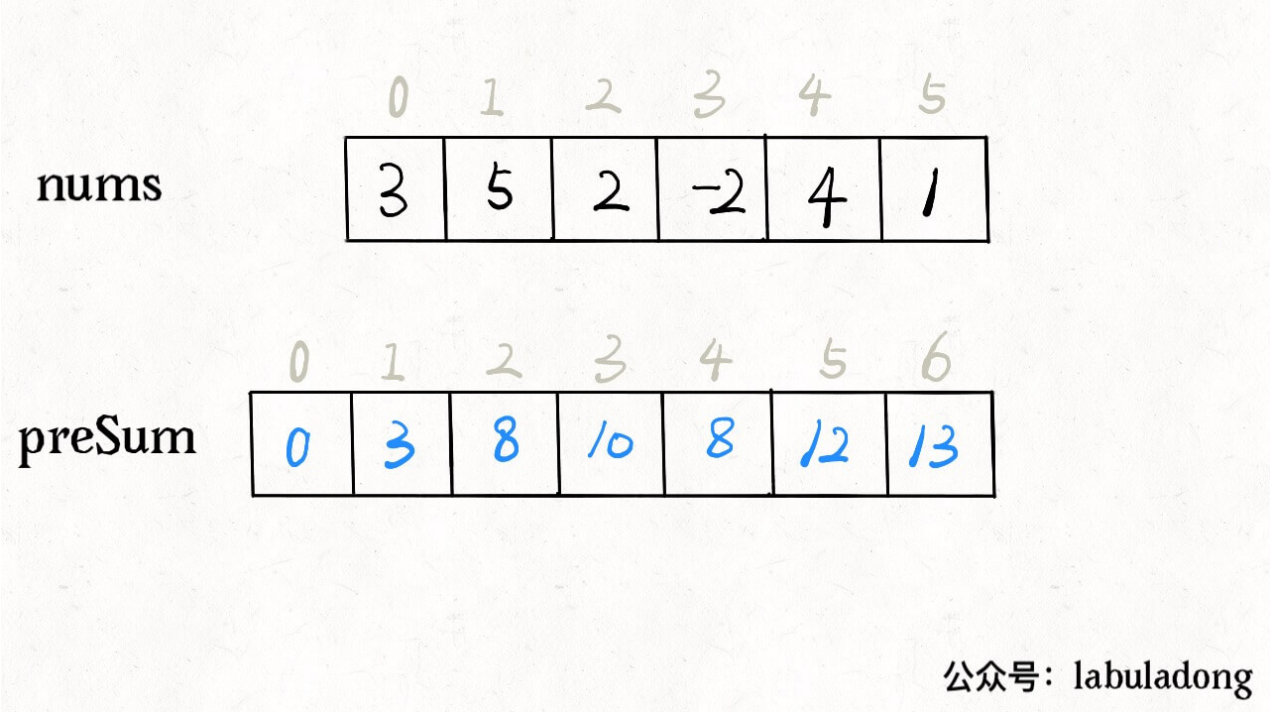
关键是，**如何快速得到某个子数组的和呢**，比如说给你一个数组 `nums`，让你实现一个接口 `sum(i, j)`，这个接口要返回 `nums[i..j]` 的和，而且会被多次调用，你怎么实现这个接口呢？

因为接口要被多次调用，显然不能每次都去遍历 `nums[i..j]`，有没有一种快速的方法在 O(1) 时间内算出 `nums[i..j]` 呢？这就需要**前缀和**技巧了。

一、什么是前缀和

前缀和的思路是这样的，对于一个给定的数组 `nums`，我们额外开辟一个前缀和数组进行预处理：

```
int n = nums.length;
// 前缀和数组
int[] preSum = new int[n + 1];
preSum[0] = 0;
for (int i = 0; i < n; i++)
    preSum[i + 1] = preSum[i] + nums[i];
```



这个前缀和数组 `preSum` 的含义也很好理解，`preSum[i]` 就是 `nums[0..i-1]` 的和。那么如果我们想求 `nums[i..j]` 的和，只需要一步操作 `preSum[j+1]-preSum[i]` 即可，而不需要重新去遍历数组了。

回到这个子数组问题，我们想求有多少个子数组的和为 k，借助前缀和技巧很容易写出一个解法：

```
int subarraySum(int[] nums, int k) {
    int n = nums.length;
    // 构造前缀和
    int[] sum = new int[n + 1];
    sum[0] = 0;
    for (int i = 0; i < n; i++)
        sum[i + 1] = sum[i] + nums[i];

    int ans = 0;
    // 穷举所有子数组
    for (int i = 1; i <= n; i++)
        for (int j = 0; j < i; j++)
            // sum of nums[j..i-1]
```

```
            if (sum[i] - sum[j] == k)
                ans++;

    return ans;
}
```

这个解法的时间复杂度 `O(N^2)` 空间复杂度 `O(N)`，并不是最优的解法。不过通过这个解法理解了前缀和数组的工作原理之后，可以使用一些巧妙的办法把时间复杂度进一步降低。

二、优化解法

前面的解法有嵌套的 for 循环：

```
for (int i = 1; i <= n; i++)
    for (int j = 0; j < i; j++)
        if (sum[i] - sum[j] == k)
            ans++;
```

第二层 for 循环在干嘛呢？翻译一下就是，**在计算，有几个 `j` 能够使得 `sum[i]` 和 `sum[j]` 的差为 k**。每找到一个这样的 `j`，就把结果加一。

我们可以把 if 语句里的条件判断移项，这样写：

```
if (sum[j] == sum[i] - k)
    ans++;
```

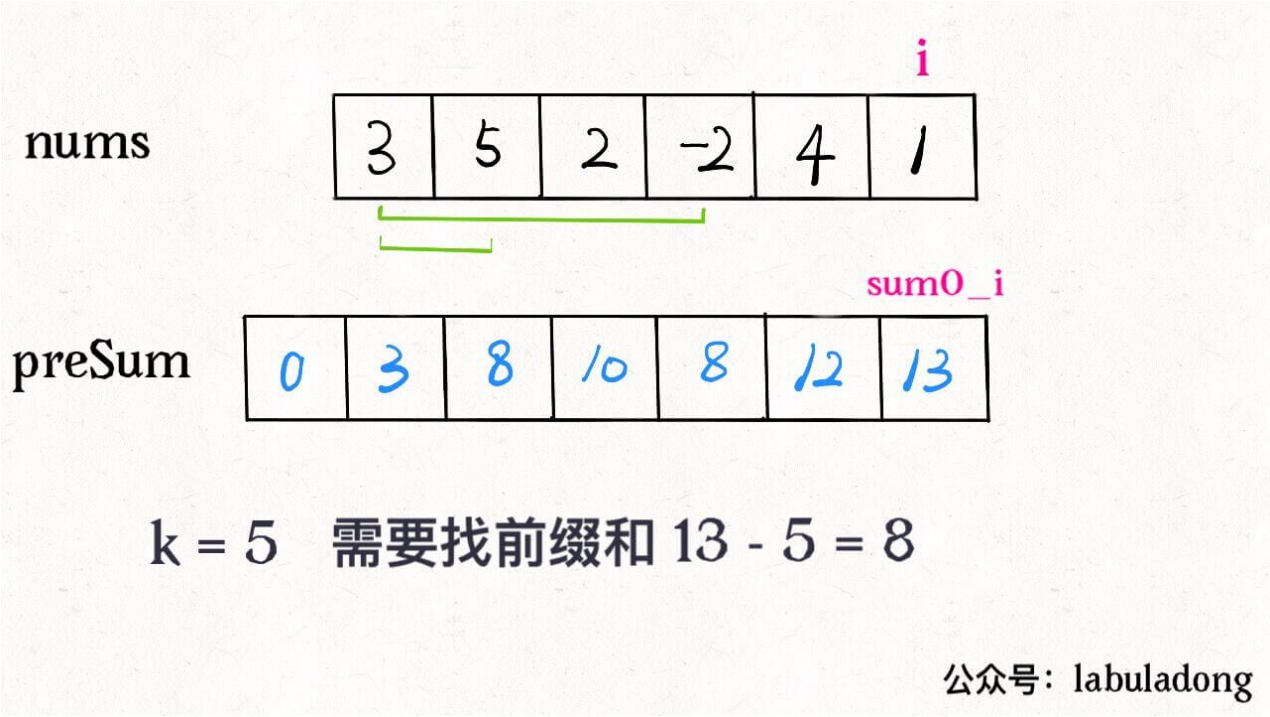
优化的思路是：**我直接记录下有几个 `sum[j]` 和 `sum[i] - k` 相等，直接更新结果，就避免了内层的 for 循环**。我们可以用哈希表，在记录前缀和的同时记录该前缀和出现的次数。

```
int subarraySum(int[] nums, int k) {
    int n = nums.length;
    // map：前缀和 -> 该前缀和出现的次数
    HashMap<Integer, Integer>
        preSum = new HashMap<>();
    // base case
    preSum.put(0, 1);

    int ans = 0, sum0_i = 0;
    for (int i = 0; i < n; i++) {
        sum0_i += nums[i];
        // 这是我们想找的前缀和 nums[0..j]
        int sum0_j = sum0_i - k;
        // 如果前面有这个前缀和，则直接更新答案
        if (preSum.containsKey(sum0_j))
            ans += preSum.get(sum0_j);
        // 把前缀和 nums[0..i] 加入并记录出现次数
        preSum.put(sum0_i,
            preSum.getOrDefault(sum0_i, 0) + 1);
```

```
    }
    return ans;
}
```

比如说下面这个情况，需要前缀和 8 就能找到和为 k 的子数组了，之前的暴力解法需要遍历数组去数有几个 8，而优化解法借助哈希表可以直接得知有几个前缀和为 8。



这样，就把时间复杂度降到了 `O(N)`，是最优解法了。

三、总结

前缀和不难，却很有用，主要用于处理数组区间的问题。

比如说，让你统计班上同学考试成绩在不同分数段的百分比，也可以利用前缀和技巧：

```
int[] scores; // 存储着所有同学的分数
// 试卷满分 150 分
int[] count = new int[150 + 1]
// 记录每个分数有几个同学
for (int score : scores)
    count[score]++
// 构造前缀和
for (int i = 1; i < count.length; i++)
    count[i] = count[i] + count[i-1];
```

这样，给你任何一个分数段，你都能通过前缀和相减快速计算出这个分数段的人数，百分比也就很容易计算了。

但是，稍微复杂一些的算法问题，不止考察简单的前缀和技巧。比如本文探讨的这道题目，就需要借助前缀和的思路做进一步的优化，借助哈希表去除不必要的嵌套循环。可见对题目的理解和细节的分析能力对于算法的优化是至关重要的。

希望本文对你有帮助。

刷算法，学套路，认准 labuladong，公众号和 在线电子书 持续更新最新文章。

本小抄即将出版，微信扫码关注公众号，后台回复「小抄」限时免费获取，回复「进群」可进刷题群一起刷题，带你搞定 LeetCode。

==其他语言代码==