涉及到子数组、求和问题，我们一定要想到前缀和，因为前缀和可以有效地节省求和时间

今天我们来说一下刷题时经常用到的前缀和思想，前缀和思想和滑动窗口会经常用在求子数组和子串问题上，当我们遇到此类问题时，则应该需要想到此类解题方式

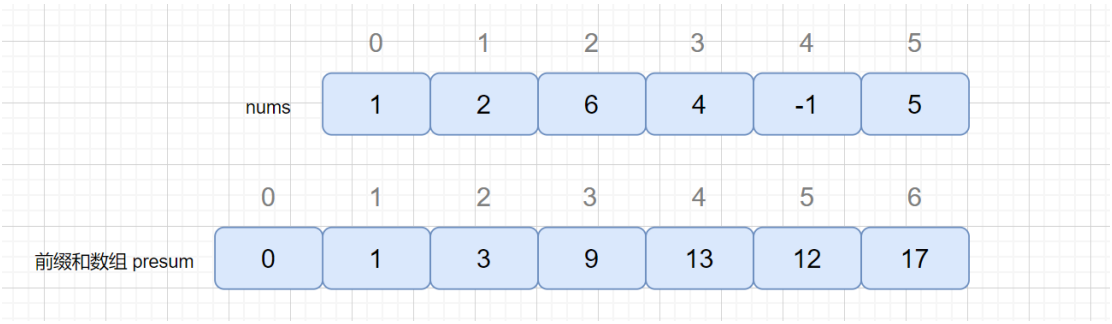
区分是用前缀和还是用滑动窗口的关键是：元素是否有负数。有负数则使用前缀和，没负数则使用滑动窗口

为什么要使用前缀和而不是滑动窗口

https://leetcode.cn/problems/subarray-sum-equals-k/solutions/247577/bao-li-jie-fa-qian-zhui-he-qian-zhui-he-you-hua-ja/

下面我们先来了解一下什么是前缀和。

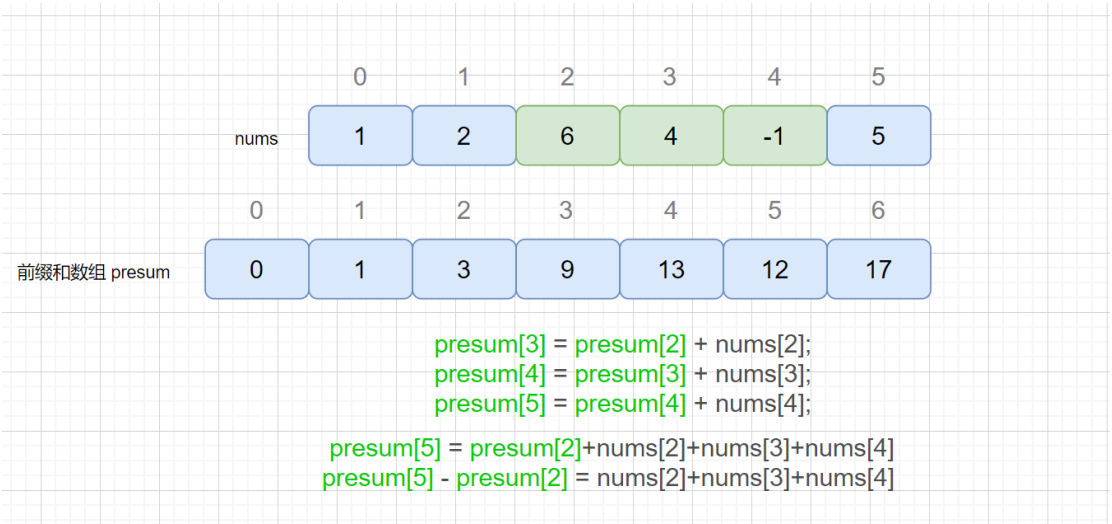
前缀和其实我们很早之前就了解过的，我们求数列的和时，Sn = a1+a2+a3+...an; 此时 Sn 就是数列的前 n 项和。例 S5 = a1 + a2 + a3 + a4 + a5; S2 = a1 + a2。所以我们完全可以通过 S5-S2 得到 a3+a4+a5 的值，这个过程就和我们做题用到的前缀和思想类似。我们的前缀和数组里保存的就是前 n 项的和。见下图



相当于前缀和数组多定义一个位置，这样剩下的位置表示原数组前 n 位的和

我们通过前缀和数组保存前 n 位的和，presum[1]保存的就是 nums 数组中前 1 位的和，也就是 presum[1] = nums[0], presum[2] = nums[0] + nums[1] = presum[1] + nums[1]. 依次类推，所以我们通过前缀和数组可以轻松得到每个区间的和。

例如我们需要获取 nums[2] 到 nums[4] 这个区间的和，我们则完全根据 presum 数组得到，是不是有点和我们之前说的字符串匹配算法中 BM,KMP 中的 next 数组和 suffix 数组作用类似。那么我们怎么根据 presum 数组获取 nums[2] 到 nums[4] 区间的和呢？见下图



也就是 **presum[5] – presum[2] = nums[2] + nums[3] + nums[4]**

获取 nums[2] 到 nums[4] 区间的和就相当于以 1 为起始 index 的数组中第三个数到第五个数之间的和。也就是前五个数的和(presum[5])减去前两个数的和(presum[2])。也就是用 presum[5] – presum[2]

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。

构造前缀和数组:

```
for (int i = 0; i < nums.length; i++) {
    presum[i+1] = nums[i] + presum[i];
}
```

好啦，我们已经了解了前缀和的解题思想了，我们可以通过上面这段代码得到我们的前缀和数组，非常简单