## 题目地址（Number of Operations to Decrement Target to Zero）

https://binarysearch.com/problems/Number-of-Operations-to-Decrement-Target-to-Zero

## 入选理由

1. 这道题太常见了，以至于力扣就有一个类似的 。https://leetcode-cn.com/problems/maximum-points-you-can-obtain-from-cards/
2. 反向思考技巧

## 标签

- 滑动窗口

## 难度

- 中等

## 题目描述

You are given a list of positive integers nums and an integer target. Consider an operation where we remove a number v from either the front or the back of nums and decrement target by v.

Return the minimum number of operations required to decrement target to zero. If it's not possible, return -1.

Constraints

n ≤ 100,000 where n is the length of nums Example 1 Input nums = [3, 1, 1, 2, 5, 1, 1] target = 7 Output 3 Explanation We can remove 1, 1 and 5 from the back to decrement target to zero.

Example 2 Input nums = [2, 4] target = 7 Output -1 Explanation There's no way to decrement target = 7 to zero.

## 前置知识

- 二分法
- 滑动窗口

# 二分法

## 思路

这道题的意思是给你一个数组，你只可以移除数组两端的数。求最小移除次数，使得移除的 数字和为 target。

我们可以反向思考，删除和为 target 的若干数字等价于保留若干和为 sum(A) - target 的数。这样问题就转化为 **求连续子数组和为 sum(A) - target 的最长子数组**。这种问 题可以使用滑动窗口来解决。

> 注意抽象后的问题有“连续”关键字，就应该想到可能会用到滑动窗口优化。具体能不能用 再根据题目信息做二次判断。

这种反向思考技巧很常见，通常是用于对比的操作上。

比如：

- 删除子集 A 的就是保留子集 A 的补集。
- 加密的反向就是解密。 比如 **5302. 加密解密字符串** 的解密等价：先将 dictionary 加密统计频率 fraq，然后返回 word2 在 fraq 的频次即 可。
- 从起点到终点的反向就是从终点到起点。比如 **LCP 20. 快速公交** 不妨从 target 出 发进行思考。
- 。。。

## 代码

代码支持：Python3， CPP

Python3 Code:

```
class Solution:
    def solve(self, A, target):
        if not A and not target: return 0
        target = sum(A) - target
        ans = len(A) + 1
        i = t = 0

        for j in range(len(A)):
            t += A[j]
            while i <= j and t > target:
                t -= A[i]
                i += 1
            if t == target: ans = min(ans, len(A) - (j - i + 1))
        return -1 if ans == len(A) + 1 else ans
```

CPP Code:

```cpp
int solve(vector<int>& nums, int target) {
    // 双指针
    int N = nums.size();
    int newTarget = accumulate(nums.begin(), nums.end(), 0) - target;
    if (newTarget == 0) return N;
    int curSum = 0;
    int maxLen = 0;
    int left = 0;  // left: 滑动窗口左边界, i: 滑动窗口右边界right
    for (int i = 0; i < N; i++)
    {
        curSum += nums[i];
        while (curSum >= newTarget && i >= left) // 寻找一个新的和为newTarget的滑动窗口区间
        {
            if (curSum == newTarget) // 当找到的新的和为newTarget的滑动窗口区间更长时, 更新其长度
                maxLen = max(maxLen, i - left + 1);

            curSum -= nums[left]; // 扔掉滑动窗口最左侧的数
            left++;
        }
    }
    return maxLen == 0 ? -1 : N - maxLen;
}
```

**复杂度分析**

令 n 为数组长度。

- 时间复杂度：$O(n)$
- 空间复杂度：1

力扣的小伙伴可以**关注我**，这样就会第一时 间收到我的动态啦~

以上就是本文的全部内容了。大家对此有何看法，欢迎给我留言，我有时间都会一一查看回 答。更多算法套路可以访问我的 LeetCode 题解仓库： https://github.com/azl397985856/leetcode 。 目前已经 40K star 啦。大家也可以关 注我的公众号《力扣加加》带你啃下算法这块硬骨头。

上一页 下一页