494. 目标和

入选理由

1. 和昨天题目很像，你还会么？

题目地址（494. 目标和）

https://leetcode-cn.com/problems/target-sum/

题目描述

给定一个非负整数数组，a1, a2, ..., an, 和一个目标数，target。现在你有两个符号 + 和 -。对于数组中的任意一个整数，你都可以从 + 或 -中选择一个符号添加在前面。

返回可以使最终数组和为目标数 target 的所有添加符号的方法数。

```
示例：

输入：nums: [1, 1, 1, 1, 1], target: 3
输出：5
解释：

-1+1+1+1+1 = 3
+1-1+1+1+1 = 3
+1+1-1+1+1 = 3
+1+1+1-1+1 = 3
+1+1+1+1-1 = 3

一共有5种方法让最终目标和为3。
```

前置知识

- 背包
- 数学

分析

题目说了数组元素都是非负的，然后我们假定**最终选择的数组中**：全正子数组和是 positive，全负子数组和是 negative，数组元素绝对值总和是 total，目标数是 target，若想符合题目要求，则必然满足如下等式：

$$\text{positive} + \text{negative} = \text{target}$$

$$\text{positive} - \text{negative} = \text{total}$$

则可以推出：

$$\text{positive} = \frac{(\text{target} + \text{total})}{2}$$

那么我们来开始抽象：

1. 数组的元素就是背包的物体
2. 元素大小就是物体重量，默认 1 为所有元素价值。
3. 包大小就是上述推出的 positive
4. 每个元素只能用一次

这就是 01 背包，目标是求出恰好装满背包的所有方案数目，可以忽略价值，不难写出如下代码。需要注意的是一些 edge case，具体参考下方代码，edge case 均在代码顶部进行了处理。

代码

代码支持：Java，Python3,JS

Java Code:

```
public int findTargetSumWays(int[] nums, int target) {

    int sum = 0;
    for (int num : nums)
        sum += num;

    if (sum < Math.abs(target))
        return 0;

    if (((sum + target) & 1) == 1)
        return 0;

    sum = (sum + target) / 2;
    int[] dp = new int[sum + 1];
    dp[0] = 1;

    for (int i = 0; i < nums.length; i++)
```

```
        for (int j = sum; j >= nums[i]; j--)
            dp[j] = dp[j] + dp[j - nums[i]];

    return dp[sum];
}
```

Python3 Code：

```
class Solution:
    def findTargetSumWays(self, nums, target) -> bool:
        t = sum(nums) + target
        if t % 2:
            return 0
        t = t // 2

        dp = [0] * (t + 1)
        dp[0] = 1

        for i in range(len(nums)):
            for j in range(t, nums[i] - 1, -1):
                dp[j] += dp[j - nums[i]]
        return dp[-1]
```

JS Code:

```
var findTargetSumWays = function (nums) {
  const sum = nums.reduce((a, b) => a + b, 0);
  let t = sum + target;
  if (t % 2) return 0;
  t = Math.floor(t / 2);
  const dp = Array(t + 1).fill(0);
  dp[0] = 1;
  for (const n of nums) {
    for (let i = t; i >= n; i--) {
      dp[i] += dp[i - n];
    }
  }
  return dp[t];
};
```

复杂度分析

令 total 为元素总和，negative 为元素个数

- 时间复杂度：$O(\frac{negative*(total+target)}{2})$

- 空间复杂度：$O(\frac{total+target}{2})$

扩展

下面是二维 dp 的做法，虽然不推荐，但是可以帮助看不懂上面滚动数组优化解法的胖友理解：

```
class Solution:
    def solve(self, nums, target):
        if (sum(nums) + target) % 2 == 1: return 0
        t = (sum(nums) + target) // 2
        dp = [[0] * (len(nums) + 1) for _ in range(t + 1)]
        dp[0][0] = 1
        for i in range(t + 1):
            for j in range(1, len(nums) + 1):
                dp[i][j] = dp[i][j-1]
                if i - nums[j-1] >= 0: dp[i][j] += dp[i - nums[j-1]][j-1]
        return dp[-1][-1]
```

另外也可使用递归来写，简单直接。

```
class Solution:
   def solve(self, nums, target):
       @lru_cache(None)
       def f(i, cur_sum):
           if i == len(nums):
               if target == cur_sum:
                   return 1
               return 0

           return f(i + 1, cur_sum + nums[i]) + f(i + 1, cur_sum - nums[i])

       return f(0, 0)
```

上一页 下一页