

Solution Review: Problem Challenge 2

We'll cover the following ^

- Target Sum (hard)
- *
- Example 1:
- Example 2:
- Solution
 - Code
 - Time and Space complexity
 - Space Optimized Solution

Target Sum (hard)

You are given a set of positive numbers and a target sum 'S'. Each number should be assigned either a '+' or '-' sign. We need to find the total ways to assign symbols to make the sum of the numbers equal to the target 'S'.

Example 1: #

Input: {1, 1, 2, 3}, S=1

Output: 3

Explanation: The given set has '3' ways to make a sum of '1': {+1-1-2+3} & {-1+1-2+3} & {+1+1+2-3}

Example 2: #

Input: {1, 2, 7, 1}, S=9

Output: 2

Explanation: The given set has '2' ways to make a sum of '9': {+1+2+7-1} & {-1+2+7+1}

Solution

This problem follows the **0/1 Knapsack pattern** and can be converted into Count of Subset Sum (<https://www.educative.io/collection/page/5668639101419520/5671464854355968/4874044023242752/>). Let's dig into this.

We are asked to find two subsets of the given numbers whose difference is equal to the given target 'S'. Take the first example above. As we saw, one solution is {+1-1-2+3}. So, the two subsets we are asked to find are {1, 3} & {1, 2} because,

$$(1 + 3) - (1 + 2) = 1$$

Now, let's say 'Sum(s1)' denotes the total sum of set 's1', and 'Sum(s2)' denotes the total sum of set 's2'. So the required equation is:

$$\text{Sum}(s1) - \text{Sum}(s2) = S$$

This equation can be reduced to the subset sum problem. Let's assume that 'Sum(num)' denotes the total sum of all the numbers, therefore:

$$\text{Sum}(s1) + \text{Sum}(s2) = \text{Sum}(\text{num})$$

Let's add the above two equations:





$$\begin{aligned} \Rightarrow \text{Sum}(s1) - \text{Sum}(s2) + \text{Sum}(s1) + \text{Sum}(s2) &= S + \text{Sum}(\text{num}) \\ \Rightarrow 2 * \text{Sum}(s1) &= S + \text{Sum}(\text{num}) \\ \Rightarrow \text{Sum}(s1) &= (S + \text{Sum}(\text{num})) / 2 \end{aligned}$$

Which means that one of the set 's1' has a sum equal to $(S + \text{Sum}(\text{num})) / 2$. This essentially converts our problem to: "Find the count of subsets of the given numbers whose sum is equal to $(S + \text{Sum}(\text{num})) / 2$ "

Code #

Let's take the dynamic programming code of Count of Subset Sum

(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/4874044023242752/>) and extend it to solve this problem:

 Java	 Python3	 C++	 JS
<pre>1 def find_target_subsets(num, s): 2 totalSum = sum(num) 3 4 # if 's + totalSum' is odd, we can't find a subset with sum equal to '(s + totalSum) / 2 5 if totalSum < s or (s + totalSum) % 2 == 1: 6 return 0 7 8 return count_subsets(num, (s + totalSum) // 2) 9 10 11 # this function is exactly similar to what we have in 'Count of Subset Sum' problem. 12 def count_subsets(num, s): 13 n = len(num) 14 dp = [[0 for x in range(s+1)] for y in range(n)] 15 16 # populate the sum = 0 columns, as we will always have an empty set for zero sum 17 for i in range(0, n): 18 dp[i][0] = 1 19 20 # with only one number, we can form a subset only when the required sum is 21 # equal to the number 22 for s in range(1, s+1): 23 dp[0][s] = 1 if num[0] == s else 0 24 25 # process all subsets for all sums 26 for i in range(1, n): 27 for s in range(1, s+1): 28 dp[i][s] = dp[i-1][s]</pre>			



Time and Space complexity

The above solution has time and space complexity of $O(N * S)$, where 'N' represents total numbers and 'S' is the desired sum.

We can further improve the solution to use only $O(S)$ space.

Space Optimized Solution

Here is the code for the space-optimized solution, using only a single array:

Java

Python3

C++

JS JS

```
1
2 def find_target_subsets(num, s):
3     totalSum = sum(num)
4
5     # if 's + totalSum' is odd, we can't find a subset with sum equal to '(s + totalSum) / 2'
6     if totalSum < s or (s + totalSum) % 2 == 1:
7         return 0
8
9     return count_subsets(num, (s + totalSum) // 2)
10
11
12 # this function is exactly similar to what we have in 'Count of Subset Sum' problem
13 def count_subsets(num, sum):
14     n = len(num)
15     dp = [0 for x in range(sum+1)]
16     dp[0] = 1
17
18     # with only one number, we can form a subset only when the required sum is equal to the
19     for s in range(1, sum+1):
20         dp[s] = 1 if num[0] == s else 0
21
22     # process all subsets for all sums
23     for i in range(1, n):
24         for s in range(sum, -1, -1):
25             if s >= num[i]:
26                 dp[s] += dp[s - num[i]]
27
28     return dp[sum]
```



← Back

Next →

Problem Challenge 2

Introduction

✓ Completed

📄 Report an Issue

🔗 Ask a Question

(https://discuss.educative.io/tag/solution-review-problem-challenge-2__pattern--01-knapsack-dynamic-programming__grokking-the-coding-interview-patterns-for-coding-questions)

