Count of Subset Sum

We'll cover the following

Problem Statement#

Given a set of positive numbers, find the total number of subsets whose sum is equal to a given number ‘S’.

Example 1:#

```
Input: {1, 1, 2, 3}, S=4
Output: 3
The given set has '3' subsets whose sum is '4': {1, 1, 2}, {1, 3}, {1,
}
```

```
Note that we have two similar sets {1, 3}, because we have
r input.
```

Example 2:#

```
Input: {1, 2, 7, 1, 5}, S=9
Output: 3
The given set has '3' subsets whose sum is '9': {2, 7}, {1, 7, 1}, {1, 2
, 1, 5}
```

Try it yourself#

Try solving this question here:

Python3

```
def count_subsets(num, sum1):
  # TODO: Write - Your - Code
  return -1
```

Basic Solution#

This problem follows the **0/1 Knapsack pattern** and is quite similar to Subset Sum. The only difference in this problem is that we need to count the number of subsets, whereas in the Subset Sum we only wanted to know if there exists a subset with the given sum.

A basic brute-force solution could be to try all subsets of the given numbers to count the subsets that have a sum equal to ‘S’. So our brute-force algorithm will look like:

```
for each number 'i'
  create a new set which includes number 'i' if it does not
      process the remaining numbers and sum
  create a new set without number 'i', and recursively process the remaining
return the count of subsets who has a sum equal to 'S'
```

Code#

Here is the code for the brute-force solution:

Python3

```python
def count_subsets(num, target_sum):
  return count_subsets_recursive(num, target_sum, 0)


def count_subsets_recursive(num, target_sum, currentIndex):
  # base checks
  if target_sum == 0:
    return 1
  n = len(num)
  if n == 0 or currentIndex >= n:
    return 0

  # recursive call after selecting the number at the currentIndex
  # if the number at currentIndex exceeds the target_sum, we shouldn't proce
  sum1 = 0
  if num[currentIndex] <= target_sum:
    sum1 = count_subsets_recursive(
      num, target_sum - num[currentIndex], currentIndex + 1)

  # recursive call after excluding the number at the currentIndex
  sum2 = count_subsets_recursive(num, target_sum, currentIndex + 1)

  return sum1 + sum2


def main():
  print("Total number of subsets " + str(count_subsets([1, 1, 2, 3], 4)))
```

The time complexity of the above algorithm is exponential O , w ‘ represents the total number. The space complexity is $O(n)$, this memory is used to store the recursion stack.

Top-down Dynamic Programming with Memoization#

We can use memoization to overcome the overlapping sub-problems. We will be using a two-dimensional array to store the results of solved sub-problems. As mentioned above, we need to store results for every subset and for every possible sum.

Code#

Here is the code:

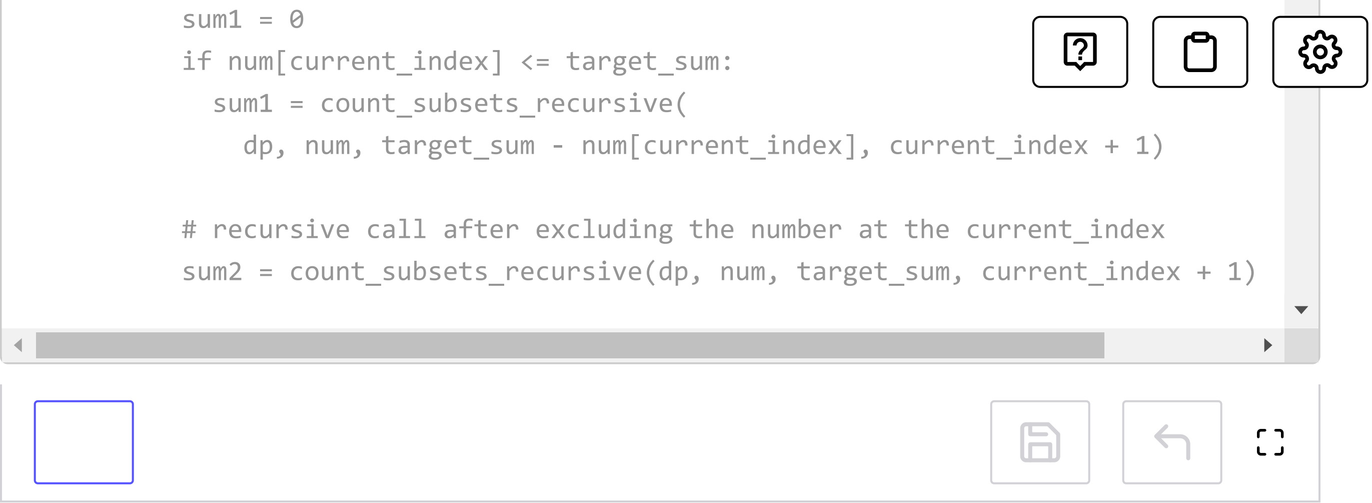
Python3

```
def count_subsets(num, target_sum):
  # create a two dimensional array for Memoization, each element is initiali
  dp = [[-1 for x in range(target_sum+1)] for y in range(len(num))]
  return count_subsets_recursive(dp, num, target_sum, 0)


def count_subsets_recursive(dp, num, target_sum, current_index):
  # base checks
  if target_sum == 0:
    return 1

  n = len(num)
  if n == 0 or current_index >= n:
    return 0

  # check if we have not already processed a similar problem
  if dp[current_index][target_sum] == -1:
    # recursive call after choosing the number at the current_index
    # if the number at current_index exceeds the sum, we shouldn't process t
```



```
    sum1 = 0
    if num[current_index] <= target_sum:
      sum1 = count_subsets_recursive(
        dp, num, target_sum - num[current_index], current_index + 1)

    # recursive call after excluding the number at the current_index
    sum2 = count_subsets_recursive(dp, num, target_sum, current_index + 1)
```

Bottom-up Dynamic Programming#

We will try to find if we can make all possible sums with every subset to populate the array `db[TotalNumbers][S+1]`.

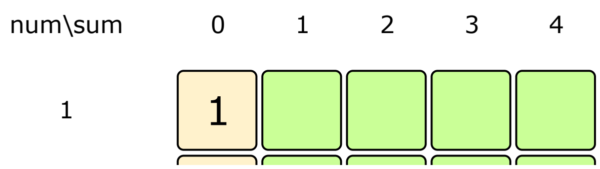
So, at every step we have two options:

1. Exclude the number. Count all the subsets without the given number up to the given sum => `dp[index-1][sum]`
2. Include the number if its value is not more than the ‘sum’. In this case, we will count all the subsets to get the remaining sum => `dp[index-1][sum-num[index]]`

To find the total sets, we will add both of the above two values:

```
dp[index][sum] = dp[index-1][sum] + dp[index-1][sum-num[index]])
```

Let’s start with our base case of size zero:



num\sum	0	1	2	3	4
1	1				

'0' sum can always be found through an empty set

1 of 12

Code#

Here is the code for our bottom-up dynamic programming approach:

Python3

```
def count_subsets(num, target_sum):
  n = len(num)
  dp = [[-1 for x in range(target_sum+1)] for y in range(n)]

  # populate the sum = 0 columns, as we will always have an empty set for ze
  for i in range(0, n):
    dp[i][0] = 1

  # with only one number, we can form a subset only when the required sum is
  # equal to its value
  for s in range(1, target_sum+1):
    dp[0][s] = 1 if num[0] == s else 0

  # process all subsets for all sums
  for i in range(1, n):
    for s in range(1, target_sum+1):
      # exclude the number
      dp[i][s] = dp[i - 1][s]
      # include the number, if it does not exceed the sum
      if s >= num[i]:
        dp[i][s] += dp[i - 1][s - num[i]]

  # the bottom-right corner will have our answer.
  return dp[n - 1][target_sum]
```

```
    def main():
```

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

Challenge#

Can we further improve our bottom-up DP solution? Can you find an algorithm that has $O(S)$ space complexity?

Show Hint

Python3

```
    def count_subsets(num, target_sum):
      # TODO: Write - Your - Code
      return -1
```

← Back Next →

Minimum Subset Sum Difference

Next

Target Sum

☑ Completed

Report an Issue