**Target Sum**

**WE'LL COVER THE FOLLOWING**

* + - [Problem Statement](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#problem-statement) \*
      * [Example 1:](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#example-1)
      * [Example 2:](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#example-2)
    - [Solution](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#solution)
      * [Code](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#code)
    - [Space Optimized Solution](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#space-optimized-solution)

**Problem Statement**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#problem-statement)

Given a set of positive numbers (non zero) and a target sum ‘S’. Each number should be assigned either a ‘+’ or ‘-’ sign. We need to find out total ways to assign symbols to make the sum of numbers equal to target ‘S’.

**Example 1:**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#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:**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#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**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#solution)

This problem follows the **0/1 Knapsack pattern** and can be converted into [Count of Subset Sum](https://www.educative.io/collection/page/5668639101419520/5633779737559040/5712536552865792/). 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:

    Sum(s1) - 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:

    Sum(s1) + Sum(s2) = Sum(num)

Let’s add the above two equations:

    => Sum(s1) - Sum(s2) + Sum(s1) + Sum(s2) = S + Sum(num)  
    => 2 \* Sum(s1) =  S + Sum(num)  
    => Sum(s1) = (S + Sum(num)) / 2

This essentially converts our problem to: “Find count of subsets of the given numbers whose sum is equal to”,

    => (S + Sum(num)) / 2

**Code**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#code)

Let’s take the dynamic programming code of [Count of Subset Sum](https://www.educative.io/collection/page/5668639101419520/5633779737559040/5712536552865792/) and extend it to solve this problem:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A)

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

    // populate the sum=0 columns, as we will always have an empty set for zero sum

    for(int i=0; i < n; i++)

      dp[i][0] = 1;

    // with only one number, we can form a subset only when the required sum is equal to the number

   // for(int s=1; s <= sum ; s++) {

     // dp[0][s] = (num[0] == s ? 1 : 0);

    //}

    // process all subsets for all sums

    for(int i=1; i < num.length; i++) {

      for(int s=1; s <= sum; s++) {

          dp[i][s] = dp[i-1][s];

          if(s >= num[i])

            dp[i][s] += dp[i-1][s-num[i]];

      }

    }

    // the bottom-right corner will have our answer.

    return dp[num.length-1][sum];

  }

  public static void main(String[] args) {

    TargetSum ts = new TargetSum();

    int[] num = {1, 1, 2, 3};

    System.out.println(ts.findTargetSubsets(num, 1));

    num = new int[]{1, 2, 7, 1};

    System.out.println(ts.findTargetSubsets(num, 9));

  }

}





RUN

SAVERESET

The above solution has time and space complexity of O(N\*S)*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)*O*(*S*) space.

**Space Optimized Solution**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A#space-optimized-solution)

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

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAOY4oy64A)

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

13

12

  // this function is exactly similar to what we have in 'Count of Subset Sum' problem.

  private int countSubsets(int[] num, int sum) {

    int n = num.length;

    int[] dp = new int[sum + 1];

    dp[0] = 1;

    // with only one number, we can form a subset only when the required sum is equal to the number

    for(int s=1; s <= sum ; s++) {

      dp[s] = (num[0] == s ? 1 : 0);

    }

    // process all subsets for all sums

    for(int i=1; i < num.length; i++) {

      for(int s=sum; s >= 0; s--) {

          if(s >= num[i])

            dp[s] += dp[s-num[i]];

      }

    }

    return dp[sum];

  }

  public static void main(String[] args) {

    TargetSum ts = new TargetSum();

    int[] num = {1, 1, 2, 3};

    System.out.println(ts.findTargetSubsets(num, 1));

    num = new int[]{1, 2, 7, 1};

    System.out.println(ts.findTargetSubsets(num, 9));

  }

  }

    return countSubsets(num, (s + totalSum) / 2);





RUN

SAVERESET

**MARK AS COMPLETED**

[**←    Back**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27pwPZ3yvG)

Count of Subset Sum

[**Next    →**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/qV6RXWME4D3)

Unbounded Knapsack

Stuck? Get help on

[**DISCUSS**](https://discuss.educative.io/c/grokking-dynamic-programming-patterns-for-coding-interviews-design-gurus/pattern-1-0-1-knapsack-target-sum)

Send feedback

14 Recommendations

Cannot edit in read-only editor (occurred 98 times)