Problem 494: Target Sum

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an integer array

nums

and an integer

target

.

You want to build an

expression

out of nums by adding one of the symbols

'+'

and

'-'

before each integer in nums and then concatenate all the integers.

For example, if

nums = [2, 1]

, you can add a

'+'

before

2

and a

'-'

before

1

and concatenate them to build the expression

"+2-1"

.

Return the number of different

expressions

that you can build, which evaluates to

target

.

Example 1:

Input:

nums = [1,1,1,1,1], target = 3

Output:

5

Explanation:

There are 5 ways to assign symbols to make the sum of nums be target 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 +1 + 1 + 1 + 1 - 1 = 3

Example 2:

Input:

nums = [1], target = 1

Output:

1

Constraints:

1 <= nums.length <= 20

0 <= nums[i] <= 1000

0 <= sum(nums[i]) <= 1000

-1000 <= target <= 1000

Code Snippets

C++:

```
class Solution {
public:
int findTargetSumWays(vector<int>& nums, int target) {

}
```

```
};
```

Java:

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

}
}
```

Python3:

```
class Solution:
def findTargetSumWays(self, nums: List[int], target: int) -> int:
```

Python:

```
class Solution(object):
def findTargetSumWays(self, nums, target):
"""
:type nums: List[int]
:type target: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} target
* @return {number}
*/
var findTargetSumWays = function(nums, target) {

};
```

TypeScript:

```
function findTargetSumWays(nums: number[], target: number): number {

};
```

C#:

```
public class Solution {
public int FindTargetSumWays(int[] nums, int target) {

}
}
```

C:

```
int findTargetSumWays(int* nums, int numsSize, int target) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun findTargetSumWays(nums: IntArray, target: Int): Int {

}
}
```

Swift:

```
class Solution {
func findTargetSumWays(_ nums: [Int], _ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_target_sum_ways(nums: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def find_target_sum_ways(nums, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function findTargetSumWays($nums, $target) {

}
}
```

Dart:

```
class Solution {
int findTargetSumWays(List<int> nums, int target) {

}
}
```

Scala:

```
object Solution {
def findTargetSumWays(nums: Array[Int], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_target_sum_ways(nums :: [integer], target :: integer) :: integer
```

```
def find_target_sum_ways(nums, target) do

end
end
```

Erlang:

```
-spec find_target_sum_ways(Nums :: [integer()], Target :: integer()) ->
integer().
find_target_sum_ways(Nums, Target) ->
.
```

Racket:

```
(define/contract (find-target-sum-ways nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Target Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int findTargetSumWays(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Target Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int findTargetSumWays(int[] nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Target Sum
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def findTargetSumWays(self, nums: List[int], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findTargetSumWays(self, nums, target):
"""
:type nums: List[int]
:type target: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Target Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @param {number} target
* @return {number}
*/
var findTargetSumWays = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Target Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findTargetSumWays(nums: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Target Sum
* Difficulty: Medium
```

```
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int FindTargetSumWays(int[] nums, int target) {

}
}
```

C Solution:

```
/*
* Problem: Target Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int findTargetSumWays(int* nums, int numsSize, int target) {

}
```

Go Solution:

```
// Problem: Target Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

```
}
```

Kotlin Solution:

```
class Solution {
fun findTargetSumWays(nums: IntArray, target: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findTargetSumWays(_ nums: [Int], _ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Target Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_target_sum_ways(nums: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def find_target_sum_ways(nums, target)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function findTargetSumWays($nums, $target) {

}
}
```

Dart Solution:

```
class Solution {
int findTargetSumWays(List<int> nums, int target) {

}
}
```

Scala Solution:

```
object Solution {
def findTargetSumWays(nums: Array[Int], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_target_sum_ways(nums :: [integer], target :: integer) :: integer
def find_target_sum_ways(nums, target) do

end
end
```

Erlang Solution:

```
-spec find_target_sum_ways(Nums :: [integer()], Target :: integer()) ->
integer().
find_target_sum_ways(Nums, Target) ->
.
```

Racket Solution:

```
(define/contract (find-target-sum-ways nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```