Target Sum

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. Now you have 2 symbols `+` and `-`. For each integer, you should choose one from `+` and `-` as its new symbol.

Find out how many ways to assign symbols to make sum of integers equal to target S.

Example 1:

```
Input: nums is [1, 1, 1, 1, 1], S is 3.
Output: 5
Explanation:

-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

There are 5 ways to assign symbols to make the sum of nums be target 3.
```

Note:

1. The length of the given array is positive and will not exceed 20.
2. The sum of elements in the given array will not exceed 1000.
3. Your output answer is guaranteed to be fitted in a 32-bit integer.

Solution 1

The recursive solution is very slow, because its runtime is exponential

The original problem statement is equivalent to:
Find a **subset** of `nums` that need to be positive, and the rest of them negative, such that the sum is equal to `target`

Let `P` be the positive subset and `N` be the negative subset
For example:
Given `nums = [1, 2, 3, 4, 5]` and `target = 3` then one possible solution is `+1-2+3-4+5 = 3`
Here positive subset is `P = [1, 3, 5]` and negative subset is `N = [2, 4]`

Then let's see how this can be converted to a subset sum problem:

```
                  sum(P) - sum(N) = target
sum(P) + sum(N) + sum(P) - sum(N) = target + sum(P) + sum(N)
                       2 * sum(P) = target + sum(nums)
```

So the original problem has been converted to a subset sum problem as follows:
Find a **subset** `P` of `nums` such that `sum(P) = (target + sum(nums)) / 2`

Note that the above formula has proved that `target + sum(nums)` must be even
We can use that fact to quickly identify inputs that do not have a solution (Thanks to @BrunoDeNadaiSarnaglia for the suggestion)
For detailed explanation on how to solve subset sum problem, you may refer to Partition Equal Subset Sum

Here is Java solution (15 ms)

```
    public int findTargetSumWays(int[] nums, int s) {
        int sum = 0;
        for (int n : nums)
            sum += n;
        return sum < s || (s + sum) % 2 > 0 ? 0 : subsetSum(nums, (s + sum) >>> 1
);
    }

    public int subsetSum(int[] nums, int s) {
        int[] dp = new int[s + 1];
        dp[0] = 1;
        for (int n : nums)
            for (int i = s; i >= n; i--)
                dp[i] += dp[i - n];
        return dp[s];
    }
```

Here is C++ solution (3 ms)

```
class Solution {
public:
    int findTargetSumWays(vector<int>& nums, int s) {
        int sum = accumulate(nums.begin(), nums.end(), 0);
        return sum < s || (s + sum) & 1 ? 0 : subsetSum(nums, (s + sum) >> 1);
    }

    int subsetSum(vector<int>& nums, int s) {
        int dp[s + 1] = { 0 };
        dp[0] = 1;
        for (int n : nums)
            for (int i = s; i >= n; i--)
                dp[i] += dp[i - n];
        return dp[s];
    }
};
```

written by yuxiangmusic original link here

Solution 2

```java
public class Solution {
    public int findTargetSumWays(int[] nums, int s) {
        int sum = 0;
        for(int i: nums) sum+=i;
        if(s>sum || s<-sum) return 0;
        int[] dp = new int[2*sum+1];
        dp[0+sum] = 1;
        for(int i = 0; i<nums.length; i++){
            int[] next = new int[2*sum+1];
            for(int k = 0; k<2*sum+1; k++){
                if(dp[k]!=0){
                    next[k + nums[i]] += dp[k];
                    next[k - nums[i]] += dp[k];
                }
            }
            dp = next;
        }
        return dp[sum+s];
    }
}
```

Please up vote if it helped, so that more people can see it. ;)

written by Chidong original link here

Solution 3

I'm quite surprised that simple DFS could pass the test since for DFS solution there are obvious a lot of overlap subproblems. So I used a map to record the intermediate result while we are walking along the recursion tree.

```
public class Solution {
    public int findTargetSumWays(int[] nums, int S) {
        if (nums == null || nums.length == 0){
            return 0;
        }
        return helper(nums, 0, 0, S, new HashMap<>());
    }
    private int helper(int[] nums, int index, int sum, int S, Map<String, Integer
> map){
        String encodeString = index + "->" + sum;
        if (map.containsKey(encodeString)){
            return map.get(encodeString);
        }
        if (index == nums.length){
            if (sum == S){
                return 1;
            }else {
                return 0;
            }
        }
        int curNum = nums[index];
        int add = helper(nums, index + 1, sum - curNum, S, map);
        int minus = helper(nums, index + 1, sum + curNum, S, map);
        map.put(encodeString, add + minus);
        return add + minus;
    }
}
```

written by butzhang original link here