0494. 目标和

ITCharge 大约 5 分钟

- 标签：数组、动态规划、回溯
- 难度：中等

题目链接

- 0494. 目标和 - 力扣

题目大意

描述：给定一个整数数组 $nums$ 和一个整数 $target$。数组长度不超过 20。向数组中每个整数前加 `+` 或 `-` 。然后串联起来构造成一个表达式。

要求：返回通过上述方法构造的、运算结果等于 $target$ 的不同表达式数目。

说明：

- $1 \le nums.length \le 20$。
- $0 \le nums[i] \le 1000$。
- $0 \le sum(nums[i]) \le 1000$。
- $-1000 \le target \le 1000$。

示例：

- 示例 1：

```py
输入：nums = [1,1,1,1,1], target = 3
输出：5
解释：一共有 5 种方法让最终目标和为 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
```

- 示例 2：

```py
输入：nums = [1], target = 1
输出：1
```

解题思路

思路 1：深度优先搜索（超时）

使用深度优先搜索对每位数字进行 `+` 或者 `-`，具体步骤如下：

1. 定义从位置 0、和为 0 开始，到达数组尾部位置为止，和为 $target$ 的方案数为 `dfs(0, 0)`。
2. 下面从位置 0、和为 0 开始，以深度优先搜索遍历每个位置。
3. 如果当前位置 i 到达最后一个位置 $size$：
    1. 如果和 cur_sum 等于目标和 $target$，则返回方案数 1。
    2. 如果和 cur_sum 不等于目标和 $target$，则返回方案数 0。
4. 递归搜索 $i + 1$ 位置，和为 $cur\_sum - nums[i]$ 的方案数。
5. 递归搜索 $i + 1$ 位置，和为 $cur\_sum + nums[i]$ 的方案数。
6. 将 4 ~ 5 两个方案数加起来就是当前位置 i、和为 cur_sum 的方案数，返回该方案数。
7. 最终方案数为 `dfs(0, 0)`，将其作为答案返回即可。

思路 1：代码

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

        def dfs(i, cur_sum):
            if i == size:
                if cur_sum == target:
                    return 1
                else:
                    return 0
            ans = dfs(i + 1, cur_sum - nums[i]) + dfs(i + 1, cur_sum + nums[i])
            return ans

        return dfs(0, 0)
```

思路 1：复杂度分析

- **时间复杂度**：$O(2^n)$。其中 n 为数组 $nums$ 的长度。
- **空间复杂度**：$O(n)$。递归调用的栈空间深度不超过 n。

思路 2：记忆化搜索

在思路 1 中我们单独使用深度优先搜索对每位数字进行 `+` 或者 `-` 的方法超时了。所以我们考虑使用记忆化搜索的方式，避免进行重复搜索。

这里我们使用哈希表 $$table$$ 记录遍历过的位置 i 及所得到的的当前和 cur_sum 下的方案数，来避免重复搜索。具体步骤如下：

1. 定义从位置 0、和为 0 开始，到达数组尾部位置为止，和为 $target$ 的方案数为 `dfs(0, 0)` 。
2. 下面从位置 0、和为 0 开始，以深度优先搜索遍历每个位置。
3. 如果当前位置 i 遍历完所有位置：
   1. 如果和 cur_sum 等于目标和 $target$，则返回方案数 1。
   2. 如果和 cur_sum 不等于目标和 $target$，则返回方案数 0。
4. 如果当前位置 i、和为 cur_sum 之前记录过（即使用 $table$ 记录过对应方案数），则返回该方案数。
5. 如果当前位置 i、和为 cur_sum 之前没有记录过，则：
   1. 递归搜索 $i + 1$ 位置，和为 $cur\_sum - nums[i]$ 的方案数。
   2. 递归搜索 $i + 1$ 位置，和为 $cur\_sum + nums[i]$ 的方案数。
   3. 将上述两个方案数加起来就是当前位置 i、和为 cur_sum 的方案数，将其记录到哈希表 $table$ 中，并返回该方案数。
6. 最终方案数为 `dfs(0, 0)` ，将其作为答案返回即可。

思路 2：代码

```py
class Solution:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        size = len(nums)
        table = dict()

        def dfs(i, cur_sum):
```

```python
        if i == size:
            if cur_sum == target:
                return 1
            else:
                return 0

        if (i, cur_sum) in table:
            return table[(i, cur_sum)]

        cnt = dfs(i + 1, cur_sum - nums[i]) + dfs(i + 1, cur_sum + nums[i])
        table[(i, cur_sum)] = cnt
        return cnt

    return dfs(0, 0)
```

思路 2：复杂度分析

- **时间复杂度**：$O(2^n)$。其中 n 为数组 $nums$ 的长度。
- **空间复杂度**：$O(n)$。递归调用的栈空间深度不超过 n。

思路 3：动态规划

假设数组中所有元素和为 sum，数组中所有符号为 `+` 的元素为 sum_x，符号为 `-` 的元素和为 sum_y。则 $target = sum\_x - sum\_y$。

而 $sum\_x + sum\_y = sum$。根据两个式子可以求出 $2 \times sum\_x = target + sum$，即 $sum\_x = (target + sum) / 2$。

那么这道题就变成了，如何在数组中找到一个集合，使集合中元素和为 $(target + sum) / 2$。这就变为了「0-1 背包问题」中求装满背包的方案数问题。

1. 定义状态

定义状态 $dp[i]$ 表示为：填满容量为 i 的背包，有 $dp[i]$ 种方法。

2. 状态转移方程

填满容量为 i 的背包的方法数来源于：

1. 不使用当前 num：只使用之前元素填满容量为 i 的背包的方法数。

2. 使用当前 num：填满容量 $i - num$ 的包的方法数，再填入 num 的方法数。

则动态规划的状态转移方程为：$dp[i] = dp[i] + dp[i - num]$。

3. 初始化

初始状态下，默认填满容量为 0 的背包有 1 种办法（什么也不装）。即 $dp[i] = 1$。

4. 最终结果

根据状态定义，最后输出 $dp[sise]$（即填满容量为 $size$ 的背包，有 $dp[size]$ 种方法）即可，其中 $size$ 为数组 $nums$ 的长度。

思路 3：代码

```py
class Solution:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        sum_nums = sum(nums)
        if abs(target) > abs(sum_nums) or (target + sum_nums) % 2 == 1:
            return 0
        size = (target + sum_num    / 2
        dp = [0 for _ in range(size + 1)]
        dp[0] = 1
        for num in nums:
            for i in range(size, num - 1, -1):
                dp[i] = dp[i] + dp[i - num]
        return dp[size]
```

思路 3：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 为数组 $nums$ 的长度。
- **空间复杂度**：$O(n)$。