Combination Sum IV

Given an integer array with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.

Example:

```
nums = [1, 2, 3]
target = 4

The possible combination ways are:
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)

Note that different sequences are counted as different combinations.

Therefore the output is 7.
```

Follow up:
What if negative numbers are allowed in the given array?
How does it change the problem?
What limitation we need to add to the question to allow negative numbers?

Credits:
Special thanks to @pbrother for adding this problem and creating all test cases.

Solution 1

Wish to learn better solutions from you guys.

```
public class Solution {
    public int combinationSum4(int[] nums, int target) {
        Arrays.sort(nums);
        int[] res = new int[target + 1];
        for (int i = 1; i < res.length; i++) {
     for (int num : nums) {
         if (num > i)
      break;
  else if (num == i)
      res[i] += 1;
  else
      res[i] += res[i-num];
     }
 }
        return res[target];
    }
}
```

written by feiyoshang original link here

Solution 2

The DP solution goes through every possible sum from 1 to target one by one. Using recursion can skip those sums that are not the combinations of the numbers in the given array. Also, there is no need to sort the array first.

```
public class Solution {
    Map<Integer, Integer> map = new HashMap<>();
    public int combinationSum4(int[] nums, int target) {
        int count = 0;
        if (nums == null || nums.length ==0 || target < 0 ) return 0;
        if ( target ==0 ) return 1;
        if (map.containsKey(target)) return map.get(target);
        for (int num: nums){
            count += combinationSum4(nums, target-num);
        }
        map.put(target, count);
        return count;
    }
}
```

written by yubad2000 original link here

Solution 3

Think about the recurrence relation first. How does the # of combinations of the `target` related to the # of combinations of numbers that are smaller than the `target`?

So we know that `target` is the sum of numbers in the array. Imagine we only need one more number to reach target, this number can be any one in the array, right? So the # of combinations of `target`, `comb[target] = sum(comb[target - nums[i]]), where 0 <= i < nums.length, and target >= nums[i]`.

In the example given, we can actually find the # of combinations of 4 with the # of combinations of 3(4 - 1), 2(4- 2) and 1(4 - 3). As a result, `comb[4] = comb[4-1] + comb[4-2] + comb[4-3] = comb[3] + comb[2] + comb[1]`.

Then think about the base case. Since if the target is 0, there is only one way to get zero, which is using 0, we can set `comb[0] = 1`.

Now we can come up with at least a recursive solution.

```
public int combinationSum4() {
    if (target == 0) {
        return 1;
    }
    int res = 0;
    for (int i = 0; i < nums.length; i++) {
        if (target >= nums[i]) {
            res += combinationSum4(nums, target - nums[i]);
        }
    }
    return res;
}
```

Now for a DP solution, we just need to figure out a way to store the intermediate results, to avoid the same combination sum being calculated many times. We can use an array to save those results, and check if there is already a result before calculation. We can fill the array with -1 to indicate that the result hasn't been calculated yet. 0 is not a good choice because it means there is no combination sum for the target.

```java
private int[] dp;

public int combinationSum4(int[] nums, int target) {
    dp = new int[target + 1];
    Arrays.fill(dp, -1);
    dp[0] = 1;
    return helper(nums, target);
}

private int helper(int[] nums, int target) {
    if (dp[target] != -1) {
        return dp[target];
    }
    int res = 0;
    for (int i = 0; i < nums.length; i++) {
        if (target >= nums[i]) {
            res += helper(nums, target - nums[i]);
        }
    }
    dp[target] = res;
    return res;
}
```

EDIT: The above solution is top-down. How about a bottom-up one?

```java
public int combinationSum4(int[] nums, int target) {
    int[] comb = new int[target + 1];
    comb[0] = 1;
    for (int i = 1; i < comb.length; i++) {
        for (int j = 0; j < nums.length; j++) {
            if (i - nums[j] >= 0) {
                comb[i] += comb[i - nums[j]];
            }
        }
    }
    return comb[target];
}
```

written by FreeTymeKiyan original link here