COUNT SUBSETS WITH SUM K

Date : / /

Given arr {1,2,2,3} target = 3
Count the no: of subsets with sum = target
Sol :- {1,2} {1,2} {3} o/p = 3

Recursion

```
int f(int ind, int sum,
      int[] nums,)
 if(sum == 0) return 1;
 if(ind == 0) return (num[0] == sum);
                num[0]==sum?1:0
 int notTake = f(ind-1, sum, nums);
 int take = 0
 if(nums[ind] <= sum)
   take = f(ind-1, sum-nums[ind]);
   return notTake + take;
 }
int solution(int[] nums, int ta
{
 int n = nums.size();
 return f(n-1, tar, nums);
```

Tabulation

```
public int f(int ind, int k,
 int[] nums, int[][] dp){
 if (k == 0) return 1;
 if (ind == 0) return
   nums[0] == k ? 0 : 1;
 int notTake = f(ind-1, k, nums, dp)
 int take = 0;
 if (nums[ind] <= k)
   take = f(ind-1, k-nums[ind], nums, dp);
   return dp[ind][k] = take + notTake;
 }
 int solution(int[] nums, int k){
 int n = nums.length;
 int[][] dp = new int[n][k+1];
 for (int[] i : dp){
   Arrays.fill(i, -1);
 }
 return f(n-1, k, nums, dp);
 }
```

Date : / /

Tabulation

```
public int subarraySum(int[] nums, int k) {
  int n = nums.length;
int[][] dp = new int[n][k+1];
for(int i=0; i<n; i++) dp[i][0]=1;
if(nums[0]<=k) dp[0][nums[0]]=1;
for(int ind=1; ind<n; ind++){
  for(int sum=0; sum<=k; sum++){
      int notpick = dp[ind-1][sum];
      int pick=0;
    if(nums[ind]<=sum) pick = dp[ind-1][sum-nums[ind]];
  dp[ind][sum] = pick+notpick;
}
}
return dp[n-1][k];
}
}
```

1 2 2 3 , 3
p c

Date : / /

Memoization

```
int n = nums.length;
int[] prev = new int[target+1];
int[] curr = new int[target+1];
prev[0] = 1;
curr[0] = 1;
if (nums[0] <= target) {
    prev[nums[0]] = 1;
}
for (int ind = 1; ind < n; ind++) {
    curr[0] = 1;
    for (int sum = 1; sum <= target; sum++) {
        int notTake = prev[sum];
        int take = 0;
        if (nums[ind] <= sum) {
            take = prev[sum - nums[ind]];
        }
        curr[sum] = notTake + take;
    }
    int[] temp = prev;
    prev = curr;
    curr = temp;
}
return prev[target];
}}
```

prev[sum]: No. of ways to make sum using elements up to previous index.

curr[sum] = No. of ways to make sum using elements up to curr index.

So ind 0-3 represent sum 0-3

prev = [1, 0, 0, 0]

curr = [1, 0, 0, 0]

nums[0] = 1 <= 3 ✓

prev[1] = 1 (there is way to make sum 1)

For sum 1

notTake = prev[1] = 1

take ⇒ nums[i] <= sum 2 <= 1 ✗

curr[1] = 1 + 0 = 1

prev = [1, 1, 0, 0]

curr = [1, 1, 0, 0]

For sum = 2

notTake = prev[2] = 0

take = 2 <= 2 ✓

take = prev[2-2] = 1

curr[2] = 0 + 1 = 1 curr = curr[1, 1, 1, 0]

sum = 3

notTake = prev[3] = 0 ✗

take = 2 <= 3 ✓

take = prev[3-3] = 1

Date : / /

- prev ⇒ Using the numbers I have seen so far, how many ways can I make each sum.

  Ex:- prev[2] = 5

  There are 5 ways to make sum = 2

- Curr ⇒ New scoreboard after checking next number.

  prev = new int [target + 1] =

  prev = [0, 0, 0, 0, 0, 0] ⇒ sum=0 way to make sum=0 (1 way); sum=1 (1 way); sum=2 (2 ways); sum=3 (3 ways) ← solution you need

  curr = [0, 0, 0, 0]

- No: of ways to make sum=0 is 1 way by taking nothing

  ∴ prev = [1, 0, 0, 0]

  curr = [1, 0, 0, 0]

if (nums[0] <= target) { prev[nums[0]] = 1; }

- We are taking only the first no & check "Can this number alone make a sum inside our target range"!

  Since nums[0] = 1 ⇒ 1 <= 3 ✓ we can make sum 1

  prev [nums[0]] = 1 ⇒ prev[1] = 1

  There is 1 way to make sum 1

Now

nums = [1, 2, 2, 3]

target = 3

prev = [1, 1, 0, 0]

curr = [1, 0, 0, 0]

1 2 2 3 , 3

Date : / /

Step 2 :- Element 2 at ind 1

At ind = 1 nums = nums[ind] = nums[1] = 2

At sum = 1

How many ways can we make sum = 1 using numbers [1, 2]

notTake / Take

If we don't take
we'll only have prev[1]
= [1]

take
check nums[ind] <= sum
2 <= 1
we cannot make sum with having higher number

curr[sum] = notTake + take = 1 + 0 = 1

∴ At prev [1, 1, 0, 0]
curr = [1, 1, 0, 0]

At sum = 2

How many ways we can make sum 2 using [1, 2]

notTake / Take

prev[sum]
prev[2] = 0
previously you have 0

nums[ind] <= sum
2 <= 2 ✓
take = prev[sum - num[ind]];
take = prev[2 - 2] } There is 1 way
take = prev[0] = 1 } to make empty

curr[sum] = notTake + take
= 0 + 1
= 1

At prev [1, 1, 0, 0] curr [1, 1, 1, 0]

Date : / /

At sum = 3 at ind 1

How many ways can we make sum = 3 using [1, 2]

notTake / Take.

prev[sum]
prev[3] = 0

nums[ind] <= 3
nums[1] = 2 2 <= 3 ✓

take = prev[sum - nums[ind]]

take = prev[3 - 2] = prev[1] = 1

How many ways did we have to make sum 1 before we added 2

curr[sum] = take + notTake
= 1 + 0
= 1

At prev[1, 1, 0, 0]. curr[1, 1, 1, 1] → 1 way to make 3 {1, 2}

1 way to make 0 [ ]

1 way to make 1 {1}

1 way to make 2 {2}

Now our prev holds new results & curr will become old one we'll overwrite it again.

At ind 2 prev = [1, 1, 1, 1]

curr = [0, 0, 0, 0]