

Grokking the Coding Interview: Patterns for Coding Questions

90% completed

 Search Course

Introduction

Pattern: Sliding Window

Pattern: Two Pointers

Pattern: Fast & Slow pointers

Pattern: Merge Intervals

Pattern: Cyclic Sort

Pattern: In-place Reversal of a LinkedList

Pattern: Tree Breadth First Search

Pattern: Tree Depth First Search

Pattern: Two Heaps

Pattern: Subsets

Pattern: Modified Binary Search

Pattern: Bitwise XOR

Pattern: Top 'K' Elements

Pattern: K-way merge

Pattern : 0/1 Knapsack ^ (Dynamic Programming)

- Introduction
- 0/1 Knapsack (medium)
- Equal Subset Sum Partition (medium)
- Subset Sum (medium)
- Minimum Subset Sum Difference (hard)
- Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 1

We'll cover the following

- Count of Subset Sum (hard)
- *
 - Example 1:
 - Example 2:
- Basic Solution
- Code
- Time and Space complexity
- Top-down Dynamic Programming with Memoization
 - Code
- Bottom-up Dynamic Programming
 - Code
 - Time and Space complexity
- Challenge

Count of Subset Sum (hard)

Given a set of positive numbers, find the total number of subsets whose sum is equal to a given number 'S'.

Example 1:

```
Input: {1, 1, 2, 3}, S=4
Output: 3
The given set has '3' subsets whose sum is '4': {1, 1, 2}, {1, 3}, {1, 3}
Note that we have two similar sets {1, 3}, because we have two '1' in our input.
```

Example 2:

```
Input: {1, 2, 7, 1, 5}, S=9
Output: 3
The given set has '3' subsets whose sum is '9': {2, 7}, {1, 7, 1}, {1, 2, 1, 5}
```

Basic Solution

This problem follows the [0/1 Knapsack pattern](#) and is quite similar to [Subset Sum](#). The only difference in this problem is that we need to count the number of subsets, whereas in [Subset Sum](#) we only wanted to know if a subset with the given sum existed.

A basic brute-force solution could be to try all subsets of the given numbers to count the subsets that have a sum equal to 'S'. So our brute-force algorithm will look like:

```
1 for each number 'i'
2 create a new set which includes number 'i' if it does not exceed 'S', and recursively
3     process the remaining numbers and sum
4 create a new set without number 'i', and recursively process the remaining numbers
5 return the count of subsets who has a sum equal to 'S'
```

Code

Here is the code for the brute-force solution:

 Java	 Python3	 C++	 JS
--	---	---	--

```
1 const countSubsets = function(num, sum) {
2     function countSubsetsRecursive(num, sum, currentIndex) {
3         // base checks
4         if (sum === 0) return 1;
5
6         if (num.length === 0 || currentIndex >= num.length) {
7             return 0;
8         }
9
10        // recursive call after selecting the number at the currentIndex
11        // if the number at currentIndex exceeds the sum, we shouldn't process this
12        let sum1 = 0;
13        if (num[currentIndex] <= sum) {
14            sum1 = countSubsetsRecursive(num, sum - num[currentIndex], currentIndex + 1);
15        }
16
17        // recursive call after excluding the number at the currentIndex
18        const sum2 = countSubsetsRecursive(num, sum, currentIndex + 1);
19        return sum1 + sum2;
20    }
21
22    return countSubsetsRecursive(num, sum, 0);
23 }
24
25 console.log(`Count of subset sum is: ---- ${countSubsets([1, 1, 2, 3], 4)}`);
26 console.log(`Count of subset sum is: ---- ${countSubsets([1, 2, 7, 1, 5], 9)}`);
```

Pattern: Topological Sort (Graph)

- Introduction
- Topological Sort (medium)
- Tasks Scheduling (medium)
- Tasks Scheduling Order (medium)
- All Tasks Scheduling Orders (hard)
- Alien Dictionary (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Miscellaneous

- Kth Smallest Number (hard)

Conclusions

- Where to Go from Here

[Mark Course as Completed](#)

RUN
SAVE
RESET
Close

Output

```
Count of subset sum is: ---> 3
Count of subset sum is: ---> 3
```

2.034s

Time and Space complexity

The time complexity of the above algorithm is exponential $O(2^n)$, where 'n' represents the total number. The space complexity is $O(n)$, this memory is used to store the recursion stack.

Top-down Dynamic Programming with Memoization

We can use memoization to overcome the overlapping sub-problems. We will be using a two-dimensional array to store the results of solved sub-problems. As mentioned above, we need to store results for every subset and for every possible sum.

Code

Here is the code:

Java
Python3
C++
JS JS

```

11     return 0;
12   }
13
14   dp[currentIndex] = dp[currentIndex] || [];
15   // check if we have not already processed a similar problem
16   if (typeof dp[currentIndex][sum] === 'undefined') {
17     // recursive call after choosing the number at the currentIndex
18     // if the number at currentIndex exceeds the sum, we shouldn't process this
19     let sum1 = 0;
20     if (num[currentIndex] <= sum) {
21       sum1 = countSubsetsRecursive(num, sum - num[currentIndex], currentIndex + 1);
22     }
23
24     // recursive call after excluding the number at the currentIndex
25     const sum2 = countSubsetsRecursive(num, sum, currentIndex + 1);
26
27     dp[currentIndex][sum] = sum1 + sum2;
28   }
29
30   return dp[currentIndex][sum];
31 }
32
33 return countSubsetsRecursive(num, sum, 0);
34 };
35
36 console.log(`Count of subset sum is: ---> ${countSubsets([1, 1, 2, 3], 4)}`); // 3
37 console.log(`Count of subset sum is: ---> ${countSubsets([1, 2, 7, 1, 5], 9)}`); // 3

```

1.631s

RUN
SAVE
RESET
Close

Output

```
Count of subset sum is: ---> 3
Count of subset sum is: ---> 3
```

Bottom-up Dynamic Programming

We will try to find if we can make all possible sums with every subset to populate the array `dp[TotalNumbers][S+1]`.

So, at every step we have two options:

1. Exclude the number. Count all the subsets without the given number up to the given sum => `dp[index-1][sum]`
2. Include the number if its value is not more than the 'sum'. In this case, we will count all the subsets to get the remaining sum => `dp[index-1][sum-num[index]]`

To find the total sets, we will add both of the above two values:

```
dp[index][sum] = dp[index-1][sum] + dp[index-1][sum-num[index]]
```

Let's start with our base case of size zero:

num\sum	0	1	2	3	4
1	1				
{1, 1}	1				
{1, 1, 2}	1				

{1,1,2,3}

1				
---	--	--	--	--

'0' sum can always be found through an empty set

1 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1				
{1,1,2}	1				
{1,1,2,3}	1				

With only one number, we can form a subset only when the required sum is equal to the number

2 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2			
{1,1,2}	1				
{1,1,2,3}	1				

sum: 1, index:1=> (dp[index-1][sum] + dp[index-1][sum - 1])

3 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1		
{1,1,2}	1				
{1,1,2,3}	1				

sum: 2, index:1=> (dp[index-1][sum] + dp[index-1][sum - 1])

4 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1	0	0
{1,1,2}	1				
{1,1,2,3}	1				

sum: 3,4, index:1=> (dp[index-1][sum] + dp[index-1][sum - 1])

5 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1	0	0
{1,1,2}	1	2			
{1,1,2,3}	1				

sum: 1, index:2=> dp[index-1][sum], as sum is less than the number at index 2 (i.e., 1 < 2)

6 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1	0	0
{1,1,2}	1	2	2		
{1,1,2,3}	1				

sum: 2, index:2=> (dp[index-1][sum] + dp[index-1][sum - 2])

7 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1	0	0
{1,1,2}	1	2	2	2	
{1,1,2,3}	1				

sum: 3, index:2=> (dp[index-1][sum] + dp[index-1][sum - 2])

8 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1	0	0
{1,1,2}	1	2	2	2	1
{1,1,2,3}	1				

sum: 4, index:2=> (dp[index-1][sum] + dp[index-1][sum - 2])

9 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1	0	0
{1,1,2}	1	2	2	2	1
{1,1,2,3}	1	2	2		

sum: 1,2, index:3=> dp[index-1][sum] , as the sum is less than the element at index '3'

10 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1	0	0
{1,1,2}	1	2	2	2	1
{1,1,2,3}	1	2	2	3	

sum: 3, index:3=> (dp[index-1][sum] + dp[index-1][sum - 3])

11 of 12

num\sum	0	1	2	3	4
1	1	1	0	0	0
{1, 1}	1	2	1	0	0
{1,1,2}	1	2	2	2	1
{1,1,2,3}	1	2	2	3	3

sum: 4, index:3=> (dp[index-1][sum] + dp[index-1][sum - 3])

12 of 12



Code #

Here is the code for our bottom-up dynamic programming approach:

Java Python3 C++ JS

```

1 let countSubsets = function(num, sum) {
2   const n = num.length;
3   const dp = Array(n)
4     .fill(0)
5     .map(() => Array(sum + 1).fill(0));
6
7   // populate the sum=0 columns, as we will always have an empty set for zero sum
  
```

```

8   for (let i = 0; i < n; i++) {
9     dp[i][0] = 1;
10    }
11
12 // with only one number, we can form a subset only when the required sum is equal to its value
13 for (let s = 1; s <= sum; s++) {
14   dp[0][s] = num[0] == s ? 1 : 0;
15 }
16
17 // process all subsets for all sums
18 for (let i = 1; i < num.length; i++) {
19   for (let s = 1; s <= sum; s++) {
20     // exclude the number
21     dp[i][s] = dp[i - 1][s];
22     // include the number, if it does not exceed the sum
23     if (s >= num[i]) {
24       dp[i][s] += dp[i - 1][s - num[i]];
25     }
26   }
27 }

```

RUN

SAVE

RESET



Close

Output

1.777s

```

Count of subset sum is: ---> 3
Count of subset sum is: ---> 3

```

Time and Space complexity

The above solution has the time and space complexity of $O(N * S)$, where 'N' represents total numbers and 'S' is the desired sum.

Challenge

Can we improve our bottom-up DP solution even further? Can you find an algorithm that has $O(S)$ space complexity?

Show Hint

Java	Python3	C++	JS
------	---------	-----	----

```

1 const countSubsets = function(num, sum) {
2   const n = num.length;
3   const dp = Array(sum + 1).fill(0);
4   dp[0] = 1;
5
6   // with only one number, we can form a subset only when the required sum is equal to its value
7   for (let s = 1; s <= sum; s++) {
8     dp[s] = num[0] == s ? 1 : 0;
9   }
10
11 // process all subsets for all sums
12 for (let i = 1; i < num.length; i++) {
13   for (let s = sum; s >= 0; s--) {
14     if (s >= num[i]) {
15       dp[s] += dp[s - num[i]];
16     }
17   }
18 }
19
20 return dp[sum];
21 };
22 console.log(`Count of subset sum is: ---> ${countSubsets([1, 1, 2, 3], 4)}`);
23 console.log(`Count of subset sum is: ---> ${countSubsets([1, 2, 7, 1, 5], 9)})`;

```

RUN

SAVE

RESET



Close

Output

1.620s

```

Count of subset sum is: ---> 3
Count of subset sum is: ---> 3

```



Explore



Tracks



My Courses



Edpresso



Refer a Friend

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#)

Back

Next

Problem Challenge 1

Problem Challenge 2

[Mark as Completed](#)

+

Create