Number factors

We'll cover the following

Problem Statement

Given a number ‘n’, implement a method to count how many possible ways there are to express ‘n’ as the sum of 1, 3, or 4.

Example 1:

```
n : 4
Number of ways = 4
Explanation: Following are the four ways we can express 'n' : {1,1,1,1}, {1,3}, {3
,1}, {4}
```

Example 2:

```
n : 5
Number of ways = 6
Explanation: Following are the six ways we can express 'n' : {1,1,1,1,1}, {1,1,3},
 {1,3,1}, {3,1,1},
{1,4}, {4,1}
```

Let’s first start with a recursive brute-force solution.

Basic Solution

For every number ‘i’, we have three option: subtract either 1, 3, or 4 from ‘i’ and recursively process the remaining number. So our algorithm will look like:

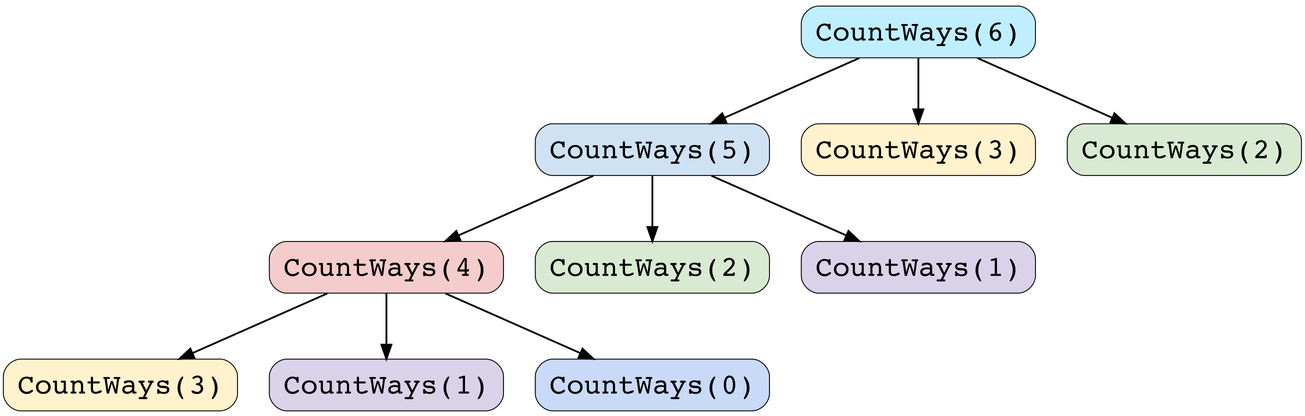
Java | JS | Python3 | C++

```
def count_ways(n):
  if n == 0:
    return 1  # base case, we don't need to subtract any thing, so there is only one way

  if n == 1:
```

```
    return 1  # we take subtract 1 to be left with zero, and that is the only way

  if n == 2:
    return 1  # we can subtract 1 twice to get zero and that is the only way

  if n == 3:
    return 2  # '3' can be expressed as {1, 1, 1}, {3}

  # if we subtract 1, we are left with 'n-1'
  subtract1 = count_ways(n - 1)
  # if we subtract 3, we are left with 'n-3'
  subtract3 = count_ways(n - 3)
  # if we subtract 4, we are left with 'n-4'
  subtract4 = count_ways(n - 4)

  return subtract1 + subtract3 + subtract4


def main():

  print(count_ways(4))
  print(count_ways(5))
  print(count_ways(6))


main()
```

The time complexity of the above algorithm is exponential $O(3^n)$. The space complexity is $O(n)$ which is used to store the recursion stack.

Let's visually draw the recursion for `CountWays(5)` to see the overlapping subproblems:



Recursion tree for calculating Fibonacci numbers

We can clearly see the overlapping subproblem pattern: `CountWays(3)`, `CountWays(2)` and `CountWays(1)` have been called twice. We can optimize this using memoization to store the results for subproblems.

Top-down Dynamic Programming with Memoization

We can use an array to store the already solved subproblems. Here is the code:

Java | JS | Python3 | C++

```python
def count_ways(n):
  dp = [0 for x in range(n+1)]
  return count_ways_recursive(dp, n)


def count_ways_recursive(dp, n):
  if n == 0:
    return 1  # base case, we don't need to subtract any thing, so there is only one way

  if n == 1:
    return 1  # we can take subtract 1 to be left with zero, and that is the only way

  if n == 2:
    return 1  # we can subtract 1 twice to get zero and that is the only way

  if n == 3:
    return 2  # '3' can be expressed as {1, 1, 1}, {3}

  if dp[n] == 0:
    # if we subtract 1, we are left with 'n-1'
    subtract1 = count_ways_recursive(dp, n - 1)
    # if we subtract 3, we are left with 'n-3'
    subtract3 = count_ways_recursive(dp, n - 3)
    # if we subtract 4, we are left with 'n-4'
    subtract4 = count_ways_recursive(dp, n - 4)

    dp[n] = subtract1 + subtract3 + subtract4

  return dp[n]


def main():

  print(count_ways(4))
  print(count_ways(5))
  print(count_ways(6))


main()
```

Bottom-up Dynamic Programming

Let’s try to populate our `dp[]` array from the above solution, working in a bottom-up fashion. As we saw in the above code, every `CountWaysRecursive(n)` is the sum of the three counts. We can use this fact to populate our array.

Code

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

Java | JS | Python3 | C++

```python
def count_ways(n):
  dp = [0 for x in range(n+1)]
```

```
    dp[0] = 1
    dp[1] = 1
    dp[2] = 1
    dp[3] = 2

    for i in range(4, n+1):
      dp[i] = dp[i - 1] + dp[i - 3] + dp[i - 4]

    return dp[n]


def main():

    print(count_ways(4))
    print(count_ways(5))
    print(count_ways(6))


main()
```

The above solution has time and space complexity of $O(n)$.

Fibonacci number pattern

We can clearly see that this problem follows the Fibonacci number pattern. However, every number in a Fibonacci series is the sum of the previous two numbers, whereas in this problem every count is a sum of previous three numbers: previous-1, previous-3, and previous-4. Here is the recursive formula for this problem:

```
CountWays(n) = CountWays(n-1) + CountWays(n-3) + CountWays(n-4), for n >= 4
```

Back
Staircase

Next
Minimum jumps to reach the end

Mark as Completed

Report an Issue

Ask a Question
(https://discuss.educative.io/tag/number-factors__pattern-3-fibonacci-numbers__grokking-dynamic-programming-patterns-for-coding-interviews)