Daily Coding Problem #12

Problem

This problem was asked by Amazon.

There exists a staircase with N steps, and you can climb up either 1 or 2 steps at a time. Given N, write a function that returns the number of unique ways you can climb the staircase. The order of the steps matters.

For example, if N is 4, then there are 5 unique ways:

- 1, 1, 1, 1
- 2, 1, 1
- 1, 2, 1
- 1, 1, 2
- 2, 2

What if, instead of being able to climb 1 or 2 steps at a time, you could climb any number from a set of positive integers X? For example, if X = {1, 3, 5}, you could climb 1, 3, or 5 steps at a time.

Solution

It's always good to start off with some test cases. Let's start with small cases and see if we can find some sort of pattern.

- N = 1: [1]
- N = 2: [1, 1], [2]

- N = 2: [1, 1], [2]
- N = 3: [1, 2], [1, 1, 1], [2, 1]
- N = 4: [1, 1, 2], [2, 2], [1, 2, 1], [1, 1, 1, 1], [2, 1, 1]

What's the relationship?

The only ways to get to N = 3, is to first get to N = 1, and then go up by 2 steps, or get to N = 2 and go up by 1 step. So f(3) = f(2) + f(1).

Does this hold for N = 4? Yes, it does. Since we can only get to the 4th step by getting to the 3rd step and going up by one, or by getting to the 2nd step and going up by two. So f(4) = f(3) + f(2).

To generalize, f(n) = f(n - 1) + f(n - 2). That's just the Fibonacci sequence!

```
def staircase(n):
    if n <= 1:
        return 1
    return staircase(n - 1) + staircase(n - 2)
```

Of course, this is really slow ($O(2^N)$) — we are doing a lot of repeated computations! We can do it a lot faster by just computing iteratively:

```
def staircase(n):
    a, b = 1, 2
    for _ in range(n - 1):
        a, b = b, a + b
    return a
```

Now, let's try to generalize what we've learned so that it works if you can take a number of steps from the set X. Similar reasoning tells us that if X = {1, 3, 5}, then our algorithm should be f(n) = f(n - 1) + f(n - 3) + f(n - 5). If n < 0, then we should return 0 since we can't start from a negative number of steps.

```
def staircase(n, X):
    if n < 0:
        return 0
    elif n == 0:

        return 1
    else:
```

```
    return sum(staircase(n - x, X) for x in X)
```

This is again, very slow ($O(|X|^N)$) since we are repeating computations again. We can use dynamic programming to speed it up.

Each entry cache[i] will contain the number of ways we can get to step i with the set X. Then, we'll build up the array from zero using the same recurrence as before:

```
def staircase(n, X):
    cache = [0 for _ in range(n + 1)]
    cache[0] = 1
    for i in range(1, n + 1):
        cache[i] += sum(cache[i - x] for x in X if i - x >= 0)
    return cache[n]
```

This now takes O(N * |X|) time and O(N) space.