Daily Coding Problem #221

Problem

This problem was asked by Zillow.

Let's define a "sevenish" number to be one which is either a power of 7, or the sum of unique powers of 7. The first few sevenish numbers are 1, 7, 8, 49, and so on. Create an algorithm to find the nth sevenish number.

Solution

A brute force solution to this problem would involve looking at consecutive integers one at a time and computing whether they are sevenish. Once we've found `n` of these, we return the last one found. To make this a little more efficient, we can use a helper function to precompute a set of sevenish numbers, by finding the totals of all subsets of the first `n` powers of 7. This way, checking whether an integer is sevenish is `O(1)`.

```
def get_sevenish_numbers(n):
    powers = [7 ** i for i in range(n)]
    totals = {0}

    for p in powers:
        totals |= {x + p for x in totals}

    return totals

def nth_sevenish_number(n):
    sevenish_numbers = get_sevenish_numbers(n)

    i = 1
    count, last_sevenish_number = 0, 0
```

```
    while count < n:
        if i in sevenish_numbers:
            count += 1
            last_sevenish_number = i
        i += 1

    return last_sevenish_number
```

Still, generating all the subsets of the first n powers of 7 is $O(2^N)$, and we must use an equivalent amount of space to store these totals.

Often when a problem involves taking powers of numbers, there is a bitwise solution, and this is no exception. Note that when we convert a number to binary, we represent it using the form $x_k * 2^k + x_{k-1} * 2^{k-1} + ... + x_0 * 2^0$. To find unique sums of powers of 7, then, we can imagine that each bit represents a power of 7 instead of 2! Let's look at the first few sevenish numbers to see how this works:

- 001 ($1 * 7^0 = 1$)
- 010 ($1 * 7^1 = 7$)
- 011 ($1 * 7^1 + 1 * 7^0 = 8$)
- 100 ($1 * 7^2 = 49$)
- 101 ($1 * 7^2 + 1 * 7^0 = 50$)

So the nth sevenish number will be the nth binary number, translated into powers of seven instead of two. This points the way to our solution: we will go through each bit of n, from least to most significant, and check if it is set. If so, we add 7^{bit_place} to our total. Once we bitshift through the entire number, we can return the total.

```
def nth_sevenish_number(n):

    answer = 0
    bit_place = 0

    while n:
        if (n & 1):
            answer += 7 ** bit_place

        n >>= 1
        bit_place += 1

    return answer
```

This algorithm is linear in the number of digits in our input and requires only constant space.