Daily Coding Problem #316

Problem

This problem was asked by Snapchat.

You are given an array of length `N`, where each element `i` represents the number of ways we can produce `i` units of change. For example, `[1, 0, 1, 1, 2]` would indicate that there is only one way to make `0`, `2`, or `3` units, and two ways of making `4` units.

Given such an array, determine the denominations that must be in use. In the case above, for example, there must be coins with value `2`, `3`, and `4`.

Solution

There are a few different ways to tackle this problem. We will first discuss the brute force method.

We know that any denomination found in our solution must have a non-zero value in the input array. To find all the candidate denomination sets, then, we can generate the power set of all combinations of these elements. For example, in the array above, `[2]`, `[3]`, `[4]`, `[2, 3]`, and so on would all be possibilities.

For each of these subsets, we can create an array that represents exactly how many ways there are of making all values between `0` and `N` using these coins. A dynamic programming approach will be helpful here. In particular, we can use the fact that the number of ways of

producing a value x is equal to the sum of all the ways of producing lower values that can reach x with a single extra coin.

Finally, we can check each of these arrays against the input array. Once we find a match, we should return the denominations responsible.

```
def get_powerset(nums):
    result = [[]]

    for x in nums:
        result.extend([subset + [x] for subset in result])

    return result

def change_combinations(coins, n):
    ways = [1] + [0] * n


    for coin in coins:
        for i in range(coin, n + 1):
            ways[i] += ways[i - coin]

    return ways

def find_denominations(array):
    nonzero = [i for i, val in enumerate(array[1:], 1) if val > 0]
    powerset = get_powerset(nonzero)

    for coins in powerset:
        ways = change_combinations(coins, len(array) - 1)
        if ways == array:
            return coins
```

There are $O(2^N)$ subsets in the power set of N, and for each of these it will take $O(N)$ time to generate the outcome array. As a result, the time complexity of this approach will be $O(N * 2^N)$. This solution will require $O(2^N)$ to store all the subsets.

Fortunately, there is a simpler method. Note that for a given index i, there are two situations in which the value of the input array at that index will be incremented:

- Coin i is one of the denominations

- Coin `i` is one of the denominations.
- Some other coin `j` is one of the denominations, and the value of `array[i - j]` is also non-zero.

Therefore, for each index `i` in the input, we can check each lower denomination `j` that is known to be part of the solution. Each time we find the value at index `i - j` to be non-zero, we have accounted for one of the way of getting to element `i`, so we can decrement the value at that index by one. If after going through all known coins, the value at index `i` is still positive, we know we must include `i` as part of our solution set.

One final note is that we must take care not to double count when iterating through our previous coins. For example, if `i = 7`, and `3` and `4` are denominations in our solution set, this only represents one way of producing `7`. To handle this case, we will add logic so that when two coins in our solution sum to an index, only the lower one will cause us to decrement the value at that index.

```
def find_denominations(array):
    coins = set()

    for i, val in enumerate(array[1:], 1):
        if val > 0:
            for coin in coins:
                if array[(i - coin)] > 0 and ((i - coin) not in coins or (i - coin) <=
coin):
                    val -= 1
            if val > 0:
                coins.add(i)

    return coins
```

The time complexity of this algorithm is `O(M * N)`, where `M` is the number of coins in our solution and `N` is the length of our input. The space complexity will be `O(M)`, since we require extra space to store the set of solution coins.