Daily Coding Problem #297

Problem

This problem was asked by Amazon.

At a popular bar, each customer has a set of favorite drinks, and will happily accept any drink among this set. For example, in the following situation, customer `0` will be satisfied with drinks `0`, 1, 3, or 6.

```
preferences = {
    0: [0, 1, 3, 6],
    1: [1, 4, 7],
    2: [2, 4, 7, 5],
    3: [3, 2, 5],
    4: [5, 8]
}
```

A lazy bartender working at this bar is trying to reduce his effort by limiting the drink recipes he must memorize. Given a dictionary input such as the one above, return the fewest number of drinks he must learn in order to satisfy all customers.

For the input above, the answer would be 2, as drinks 1 and 5 will satisfy everyone.

Solution

We have good news and bad news. The good news is that there is a greedy solution that is guaranteed to find a satisfactory set of drinks, and will often be optimal. The bad news is that this problem is actually a variation of a set cover problem, meaning any optimal solution will be NP-complete.

Let us first review the greedy solution. We can start out by rearranging our input so that the keys are drinks, and the values are the customers who like them. Following this, we create a list of all drinks, sorted by most to least popular.

Then, we iterate through this list, adding successive drinks to our solution set if they are the favorite of some yet-unsatisfied person. Once we reach the point where every customer has been served, we break out of our loop, and return the number of drinks required.

```
from collections import defaultdict

def make_drinks(preferences):
    drink_map = defaultdict(set)

    for customer, favorites in preferences.items():
        for favorite in favorites:
            drink_map[favorite].add(customer)

    drinks = sorted(drink_map, key=lambda x: len(drink_map[x]), reverse=True)

    served = set(); num_served = 0
    num_customers = len(preferences.keys())
    num_drinks = 0
    i = 0

    while num_served != num_customers:
        served |= drink_map[drinks[i]]
        if len(served) > num_served:
            num_served = len(served)
            num_drinks += 1
        i += 1

    return num_drinks
```

If the number of drinks is N, then the time complexity of this algorithm will be $O(N \log N)$, since sorting the drink list by popularity will be the most computationally expensive part. The space required will be $O(N)$, since there will be N keys in our dictionary, and no more than N elements in our set of served customers.

While this seems great, there are unfortunately cases where this will fail. Consider the

following input:

```
favorites = {
    0: [0, 3],
    1: [1, 4],
    2: [5, 6],
    3: [4, 5],
    4: [3, 5]
    5: [2, 6]
}
```

Here, our algorithm would have us choose 5 as the first drink, requiring three other selections to finish. Instead, the optimal set of drinks would be `{3, 4, 6}`.

As this problem is NP-complete, there is no shortcut to finding this optimal solution: we must search through each possible combination of favorites to find the minimal option that satisfies all customers. We can improve this process somewhat by examining smaller sets before larger ones, so that, for example, if the solution only requires three drinks, we will never check any four-drink options.

```
from itertools import combinations

def satisfies(option, preferences):
    return all(set(c).intersection(option) for c in preferences.values())

def make_drinks(preferences):
    customers = preferences.keys()
    drinks = set([x for y in preferences.values() for x in y])

    for i in range(1, len(customers) + 1):
        options = combinations(drinks, i)
        for option in options:
            if satisfies(option, preferences):
                return i
```

Checking each option will take $O(N^2)$ time in the worst case, as we must check whether there is a matching element in our candidate solution for each set of preferences. And since there are 2^N ways of forming combinations from a set, the overall time complexity will be $O(N^2 * 2^N)$.