Daily Coding Problem #252

Problem

This problem was asked by Palantir.

The ancient Egyptians used to express fractions as a sum of several terms where each numerator is one. For example, `4 / 13` can be represented as `1 / 4 + 1 / 18 + 1 / 468`.

Create an algorithm to turn an ordinary fraction `a / b`, where `a < b`, into an Egyptian fraction.

Solution

It turns out that this problem can be solved with a greedy algorithm. If we keep increasing our denominator `d`, we will eventually find a fraction `1 / d` that is less than `a / b`. Then, we can subtract this fraction from `a / b` and continue this process on the remainder.

We must be careful to use a subtraction method that keeps the numerator and denominator integers, so we don't run into floating point errors.

```
def subtract(x, y):
    a, b = x
    c, d = y
    return (a * d - b * c, b * d)

def fraction(a, b):
    denominators = []
```

```
    d = 1

    while a != 0:
        d += 1
        if (1 / d) <= (a / b):
            denominators.append(d)
            a, b = subtract((a, b), (1, d))

    return denominators
```

While this method works, it can be incredibly slow on even small input, such as `5 / 121`, for which it will eventually find the denominators `[25, 757, 763309, 873960180913, 1527612795642093385023488]`.

To improve this, instead of simply incrementing our denominator count, we can directly find the next denominator to be the smallest integer above `b / a`. This works because `(a / b)` is equal to `1 / (b / a)`, which must be greater than `1 / ceil(b / a)`. For example, for `4 / 13`, the first denominator should be `ceil(13 / 4) = 4`.

In addition, we can simplify the calculation of our new numerator by noting that since `c = 1` and we have updated `d`, `a * d - b * c` can be changed to `(-b) % a`.

```
from math import ceil

def fraction(a, b):
    denominators = []
    total = 0

    while a != 0:
        denominators.append(ceil(b / a))
        a, b = (-b) % a, b * ceil(b / a)

    return denominators
```